Tiny Transfer
User Guide and Technical Documentation

Version 1.0

May 23, 2024

Table of Contents

Purpose

Tiny Transfer makes it easier to transfer individual classic Mac OS files between computers. No server, no special operating systems, no installers, and no extensions are needed. Just a serial cable.

Tiny Transfer converts files to and from the popular BinHex 4 format, which retains resource fork and file properties when the file is posted on the Internet or stored on a modern computer. Tiny Transfer also sends and receives files over the serial port to classic Macs, modern Macs, Windows, or Linux. Serial transfer can be more convenient than saving to a disk emulator and then physically moving memory cards.

Added Benefits

Tiny Transfer retains more information than standard BinHex 4, by extending the format in a compatible manner to include:

- Creation, modification, and backup date
- Finder comment
- Extended file flags and values

Tiny Transfer also includes three extra features:

- Verifies the resource fork before and after conversion, so you aren't unintentionally archiving a corrupted copy of a file
- Synchronizes the system clock, because PRAM batteries are often removed
- Allows files to be copied and pasted from the Edit meu, which is useful to quickly transfer files to/from a Macintosh emulator to the host machine

Requirements

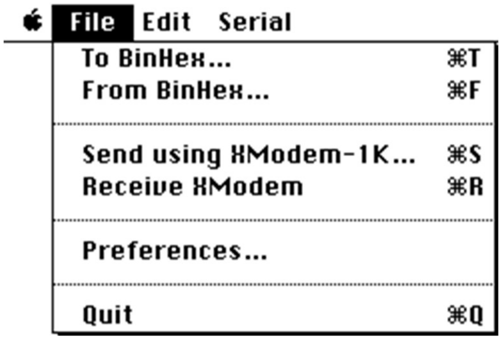
Tiny Transfer runs of all Macs and Mac OSs from System 1.0 (aka 0.97) to 9.2.2. Tiny Transfer also runs in emulators, such as Basilisk II and Mini vMac. Although Tiny Transfer runs well on Power Macintosh computers, it does not use native PowerPC code ("fat application"). Avoiding the extra code reduces the size of the Tiny Transfer and allows it to fit better on 400KB floppies and RAM/ROM disks.

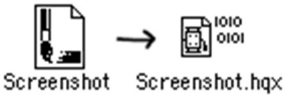
Tiny Transfer requires a minimum of 96 KB of RAM. It runs slightly faster with preferred memory of 384 KB. Beyond that, there isn't an added benefit of more memory except if you want to transfer larger files via Copy and Paste in the Edit menu.

Modem or printer port. 300 to 57600 baud. 8 data bits, no parity, 1 stop bit (8-N-1). No flow control. Does not use Ethernet, AppleTalk, Communication Toolbox, or a modem.

BinHex File Conversion

Files can be converted locally to and from BinHex 4 format by dragging the files onto the Tiny Transfer application icon or by choosing either ‘To BinHex’ or ‘From BinHex’ in Tiny Transfer’s File menu.





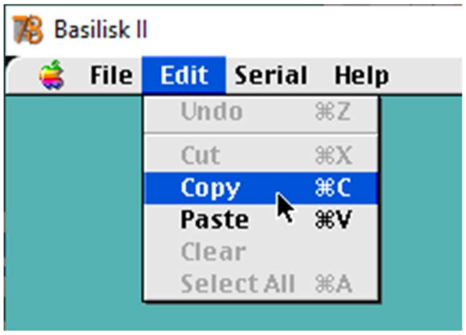
Some people may decide they only want to use Tiny Transfer for this purpose, replacing StuffIt or the BinHex application. Tiny Transfer is faster than those applications and includes extra file information, such as the Finder comment and file dates.

A great advantage of a BinHex 4 file is that it is plain text in a well-documented open format. You can open the file in a text editor to see the contents. You can even copy and paste it onto forum postings.

```
(This file must be converted with BinHex 4.0)

:$f*TEQKPH#jdCA0d,R0TG!"6594%8dP8)3%!!!!"I!#3"!&j8dP8)3!"!!!"I(*

-BA8#r`!!!"Err`d0#f*TEQKPH#jdCA0d!*jLJ!!9k`S!!!#!!*!&TSDZIkJ!N"V

rN!4849K89%*#0J%!V-H8mUc(P2)!!!'%!!!!33!!!,d!!!!jeZ1,5!#3"Vha%3!

)(23l",%&S!2,Qd`(Cq)DZ!hRBBpMDNcU%8a1$J9f!+AbpSZ`0$V%*fQQe&iYU@F

Se+T9I8H+[V6$HNH[h[HPE)9X2mZ6kH"UPN[hAPLVp[c1%D8c2V2GZcdbqJ(NJj2

1kAqRaKqd(Ad50P2(ha66QC'mQ6i[j)Q0FEMRR9Jh9!,N`HL0pbrPI3Idh'53!+0

34q-bjG"d4mX2GZi@F(MKB*%L(Dl%2*fp(NNVqm"-,Gr42M96mm&apjDb!&%'[aG

N3`iJRcXBc!DqIUc2Jd%BC[N`i`pP!LlXbq##`"m#ET!!i3rk!)Pa%-iLY,H!9Li

4lJF!!!"SH!#3"!!!:


(One more thing...)

:-3C8G'Cb!3"6!JJ36!3!`!)"3a-a16!d,6!a,6!a9$!`1M!`1M!`64-a16!d,6!

a,6!a9$!`1M!`1M!`3K-a16!d,6!a,6!a9$!`1M!`1M!`93B!N!CA!3"B!3""$e0

dG@CQ5A3J4'9XGAKPUPS#)SF!:
```

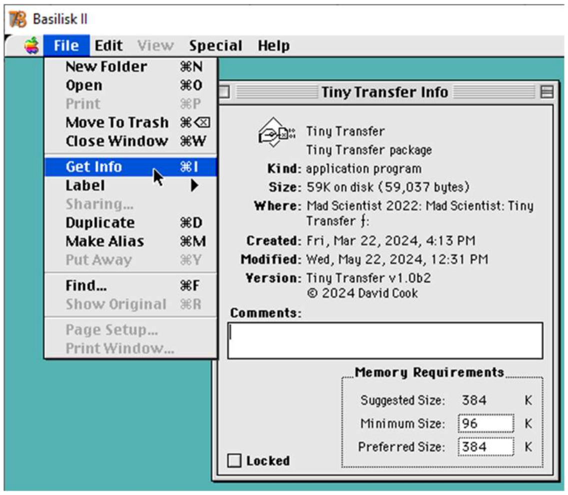
BinHex Clipboard Conversion

To make it easier to import and export files from emulators, you can choose Copy from the Edit menu to convert a file directly to the clipboard. Then, in the host machine, you can paste the BinHex text to NotePad or a forum posting, etc.



In reverse, select and copy BinHex text on the host machine. Then, in Tiny Transfer, choose Paste from the Edit menu to create the file on the disk in the emulator. The file will appear in the Tiny Transfer Files folder (learn more about this folder in the Serial Transfer section of this document).

Because files are being converted directly into memory, there must be enough memory allocated to the Tiny Transfer application to contain the BinHex text. If you are running under MultiFinder or System 7 or later, choose Get Info on Tiny Transfer in the Finder. Then, modify the preferred memory size and restart Tiny Transfer.



Tiny Transfer
Tiny Transfer package
Kind: application program
Size: 59K on disk (59,037 bytes)
Where: Mad Scientist 2022: Mad Scientist: Tiny Transfer ƒ:
Created: Fri, Mar 22, 2024, 4:13 PM
Modified: Wed, May 22, 2024, 12:31 PM
Version: Tiny Transfer v1.0b2
© 2024 David Cook
Comments:

Memory Requirements
Suggested Size: 384 K
Minimum Size: 96 K
Preferred Size: 384 K

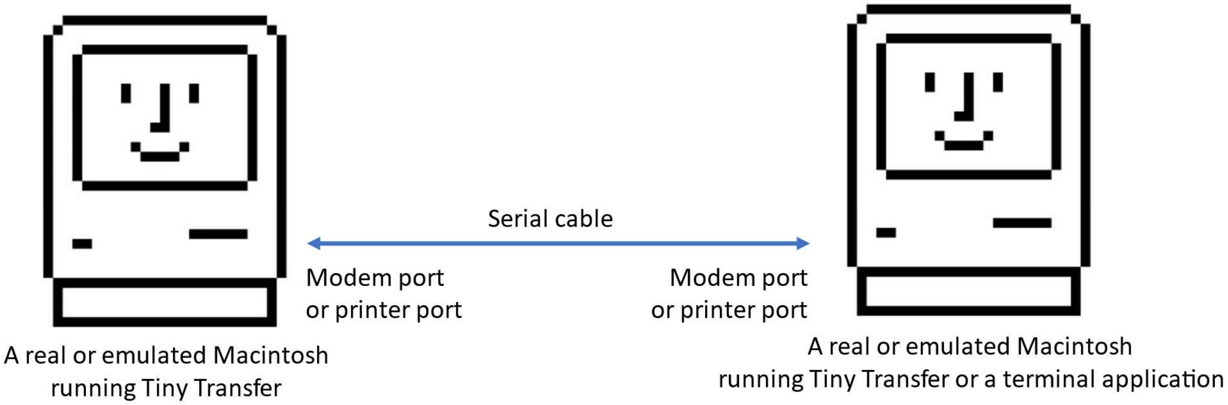
□ Locked

Note: For Mini vMac, the desk accessory or function key named ClipIn and ClipOut is necessary to import or export the clipboard to the host machine. See https://www.gryphel.com/c/minivmac/extras/ This is a limitation of MiniVMac, and not specific to Tiny Transfer.

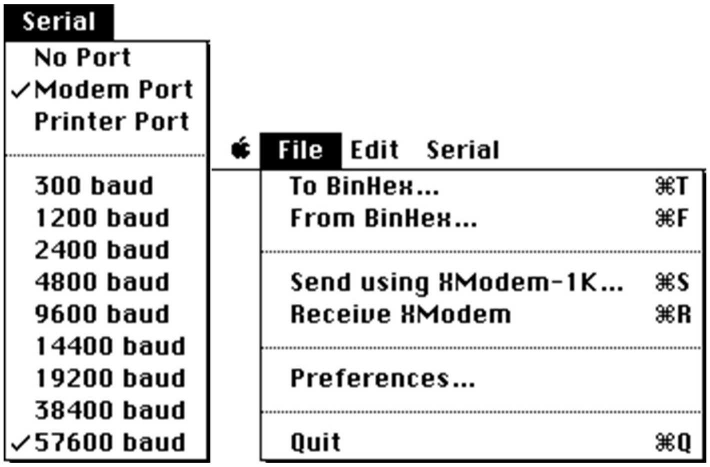
Serial Transfer

Another use of Tiny Transfer is sending files over a serial connection, peer-to-peer. The computers only need to be connected with a serial cable. No AppleTalk or special system software is used.

Simply connect two Macintosh computers via a serial cable. Run Tiny Transfer on one Macintosh and Tiny Transfer or a terminal application on the other computer.

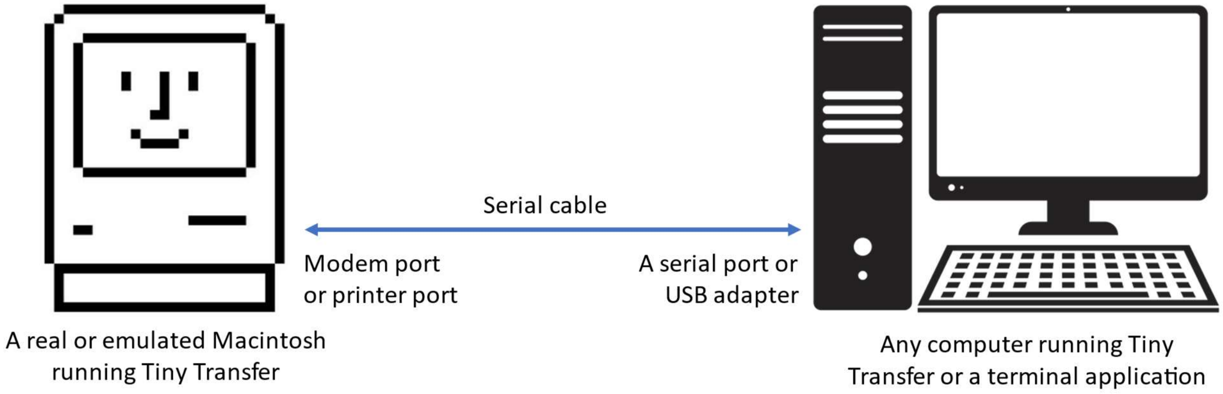

Serial cable
Modem port
or printer port
Modem port
or printer port
A real or emulated Macintosh
running Tiny Transfer
A real or emulated Macintosh
running Tiny Transfer or a terminal application

Make sure the baud rate is the same on both computers. 57600 baud delivers the fastest performance. Also, make sure the serial port is set correctly to either modem or printer, depending on where you connected the cable. On earlier Macintosh computers and systems, the modem port delivers better performance than the printer port. Unless you have a reason otherwise, use the modem port.



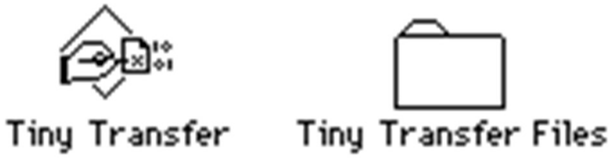
After setting up, choose “Send using XModem-1K” from Tiny Transfer’s File menu. If Tiny Transfer is also running on the other computer, it will start receiving automatically. Otherwise, choose the appropriate receive menu item in that computer’s terminal application. If Tiny Transfer does not start receiving automatically, choose ‘Receive XModem’ in the Tiny Transfer’s File menu.

This method of file transfer also works if the other computer is a Windows PC, Linux, or modern Macintosh running a terminal application or an emulator with Tiny Transfer. Modern computers lacking a DB9 serial port can use a serial-to-USB adapter.


Serial cable
Modem port or printer port
A serial port or USB adapter
A real or emulated Macintosh running Tiny Transfer
Any computer running Tiny Transfer or a terminal application

Tiny Transfer Files Folder

Files received by Tiny Transfer via serial or clipboard are stored in a folder named "Tiny Transfer Files". This folder is created automatically in the location that the Tiny Transfer application resides. If that drive is write-protected, then the folder is created in the root directory of the first writable drive.

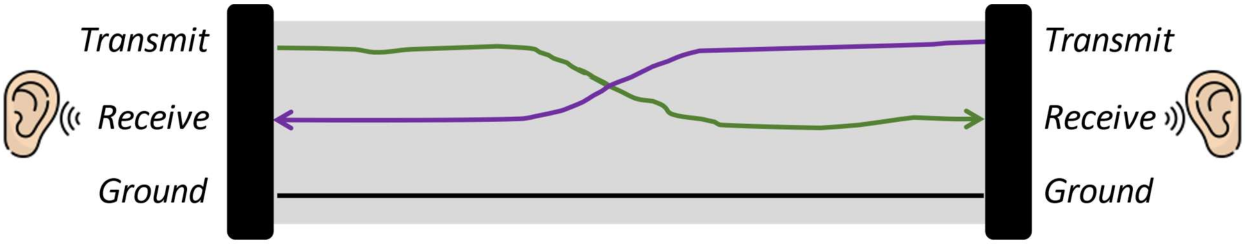


If Tiny Transfer is included in a ROM disk, then theoretically you could transfer files to a broken computer where the SCSI and floppy ports don't work. You would start a RAM disk on the broken computer and run Tiny Transfer from the ROM disk. Tiny Transfer will then receive files over the serial port and store them on the RAM disk.

Serial Cable

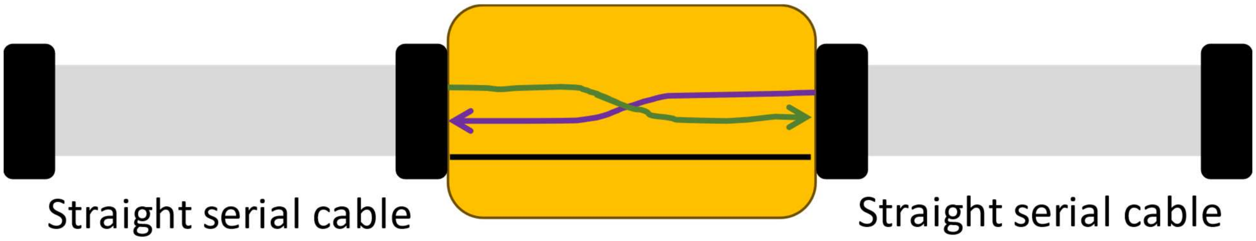
When connecting two computers to each other, you can't just use a standard straight-through serial cable. The problem is that the transmit pin of the first computer needs to connect to the receive pin of the other computer, and vice versa. There are two common ways to accomplish this.

First, you can buy or make a serial cable that has those wires crossed inside the cable. This is called a 'null modem serial cable' or a 'crosslinked serial cable'. DB9 breakout boards are available from Amazon, so you can cut off the end of an existing mini-DIN8 cable to make your own crosslinked cable.



Crosslinked serial cable

Or, second, you can use straight serial cables with a null modem adapter in between them. The "null modem" simply has the wires crosslinked inside the adapter. If you already have a straight mini-DIN8 to DB9 cable (such as Apple 590-0341-A), then attaching a null modem adapter is the easiest option. On the other end of the null-modem adapter, simply connect an appropriate cable for the other computer.



Null modem adapter

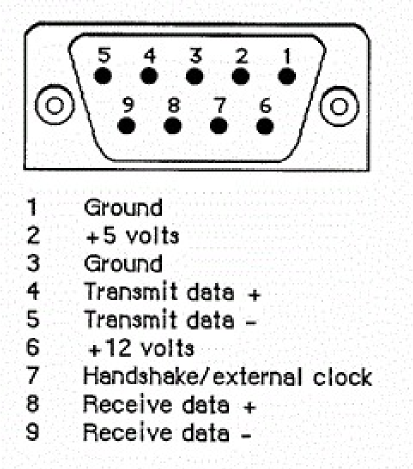
Here is a photograph of a real-world null-modem adapter between two serial cables.

The need for a null modem / crossover cable is not special for Tiny Transfer; it is true of any peer-to-peer serial connection between computers. In the old days, a pair of modems would be between the computers, performing the transmit->receive pin switchover.

By the way, if you're scrounging through piles of cables to combine a bunch together, avoid using Apple's ImageWriter DB25 (big wide connector) cable. It has the wires arranged in an unusual manner.

Macintosh 128K/512K Serial Cable

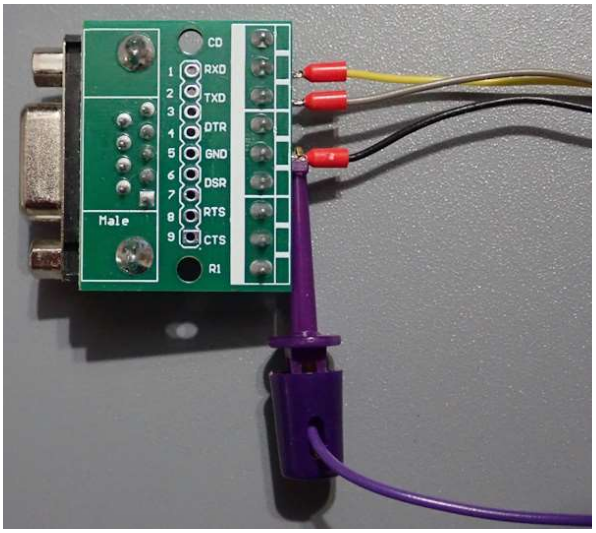
Even though they appear to be standard DB9 serial ports, the Macintosh 128K and 512K have serial ports with non-standard pinouts. You'll need to make a custom cable.



1	Ground
2	+5 volts
3	Ground
4	Transmit data +
5	Transmit data -
6	+12 volts
7	Handshake/external clock
8	Receive data +
9	Receive data -

Figure 3. Pinout for SCC Output Jacks

Fortunately, it's really easy as only four wires need to be connected. And, you can buy a DB9 breakout board and a serial cable with bare wires from Amazon or eBay. You don't even need to solder anything.



- Mac pin 1 "ground" to breakout board GND
- Mac pin 5 "transmit data -" to breakout board RXD (crosslink)
- Mac pin 8 "receive data +" to breakout board GND, according to Develop Magazine 17, page 126, March 1994. (In the image above, a purple jumper clip is used because this breakout connector couldn't fit two wires in the GND hole)
- Mac pin 9 "receive data -" to breakout board TXD (crosslink)

XModem vs ASCII File Transfer

The XModem protocol and BinHex file format both include CRC16 checks to ensure the integrity of a transfer. XModem retries any packets that are corrupted during transmission. In contrast, a raw ASCII transfer of a BinHex file is supported, but corrupted characters are not caught by a receiving terminal application. Therefore, using the XModem file transfer protocol is definitely preferrable.

There are many flavors of XModem, and many implementations of those flavors. Practically speaking, there are three types: XModem (classic), XModem-CRC, and XModem-1K. Each protocol builds on the previous. XModem (classic) sends a packet number header, 128 bytes of data, and a simple single-byte checksum. XModem-CRC replaces the checksum with a much better check (CRC16) that is two bytes long. XModem-1K expands on that and allows the packets to have 1024 bytes of data.

For small files or slow baud rates (2400 and below), Tiny Transfer automatically uses XModem-CRC. This provides more frequent progress updates to the user and doesn't waste time transferring padding in the final 1024-byte packet.

For everything else, Tiny Transfer automatically uses XModem-1K. This provides maximum speed.

Tiny Transfer automatically backs down, even all the way to XModem (classic) if a terminal application indicates issues at the start of a transfer. However, some very old terminal programs may get confused. In that case, you can choose the specific flavor of XModem you would like Tiny Transfer to start with, by choosing Preferences from the File menu.

If you are not connecting to a terminal application, and instead are just connecting Tiny Transfer to Tiny Transfer, then you needn't worry about any of this. Just leave the Tiny Transfer preference set to XModem-1K and it will take care of everything for you.

Terminal Application Setup

The preferred file transfer method is Tiny Transfer to Tiny Transfer. However, sometimes you don't have Tiny Transfer on one of the computers, or you want to transfer a file with a modern PC or Mac without an emulator.

Here are examples of how to set up some terminal applications. These examples may also help you set up terminal applications that are not specifically mentioned.

MacTerminal 2.3

1. In MacTerminal, choose Settings->Compatibility->Another Computer. This tells the terminal that it isn't talking to a modem and doesn't need to dial a number. Set the baud as desired.
2. In MacTerminal, choose Settings->File Transfer->Straight XModem.
3. In Tiny Transfer, choose Preferences->XModem-CRC. MacTerminal 2.3 does not understand XModem-1K and will never start, even after Tiny Transfer auto-downgrades. MacTerminal is likely not clearing (or ignoring) its buffer after sending the first nak, and thus MacTerminal naks bytes from the rest of the first packet until the transfer cancels.

Alternatively, you can also use Preferences->XModem (classic) in Tiny Transfer and MacTerminal will start the transfer after a brief delay. However, XModem-CRC is preferred due to the superior corruption check.

MacComCenter 1.03

MacComCenter is a pretty decent terminal application. In MacComCenter, choose:

1. Setup->Modem. Set baud, flow control off, etc.
2. Setup->File Transfer->MacBinary options: Never MacBinary
3. Setup->File Transfer->Default protocols: Xmodem 1K for send and receive

ZTerm 1.0.1

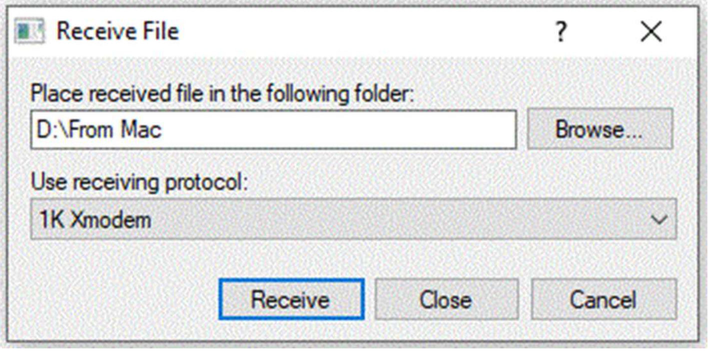
In ZTerm:

1. Settings->Connection. Set baud, flow control off, etc.
2. Settings->Transfer Options: Send XModem-1K
3. Settings->Transfer Options: Receive XModem
4. Settings->X/YModem error checking: Try CRC, fallback to Checksum
5. File->Transfer Convert->Binary Data

HyperTerminal

Here is an example of transferring files from a classic Macintosh to a Windows PC. HyperTerminal was commonly bundled on Windows PCs in the 1990s. There is also a newer version sold by a private vender.

1. File->Properties. Click the Configure button. (If Configure is disabled, first choose Call->Disconnect) Set the baud rate etc (57600, 8, none, 1, none). Make sure flow control is set to none.
2. Transfer->Receive File. In the pulldown, choose 1K Xmodem.

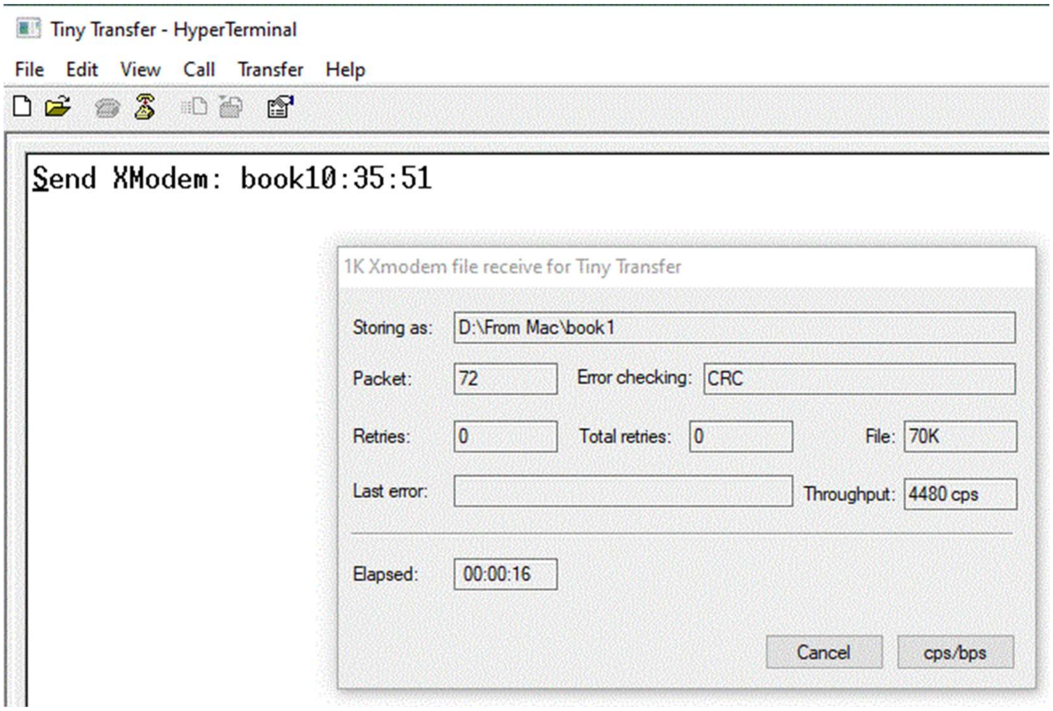


Place received file in the following folder: D:\From Mac

Use receiving protocol: 1K Xmodem

Below is an example of a file ("book1") that was successfully delivered.

At the top of the terminal screen is some overwritten text. This is because Windows expects linefeeds on text. Although this is purely aesthetic, if you will be connecting to Windows often, choose File->Preferences in Tiny Transfer and click the "Add line feeds" checkbox.


Tiny Transfer - HyperTerminal
File
Edit
View
Call
Transfer
Help
1K Xmodem file receive for Tiny Transfer
Cancel
cps/bps

```
Send XModem: book10:35:51
```

Storing as: D:\From Mac\book1

Packet: 72 Error checking: CRC

Retries: 0 Total retries: 0 File: 70K

Last error: Throughput: 4480 cps

Elapsed: 00:00:16

Synchronizing the Clock

Tiny Transfer sends its version information and the computer’s current date and time whenever Tiny Transfer opens the serial port. This usually happens when Tiny Transfer first starts up or whenever the user changes a setting in Tiny Transfer’s Serial menu.

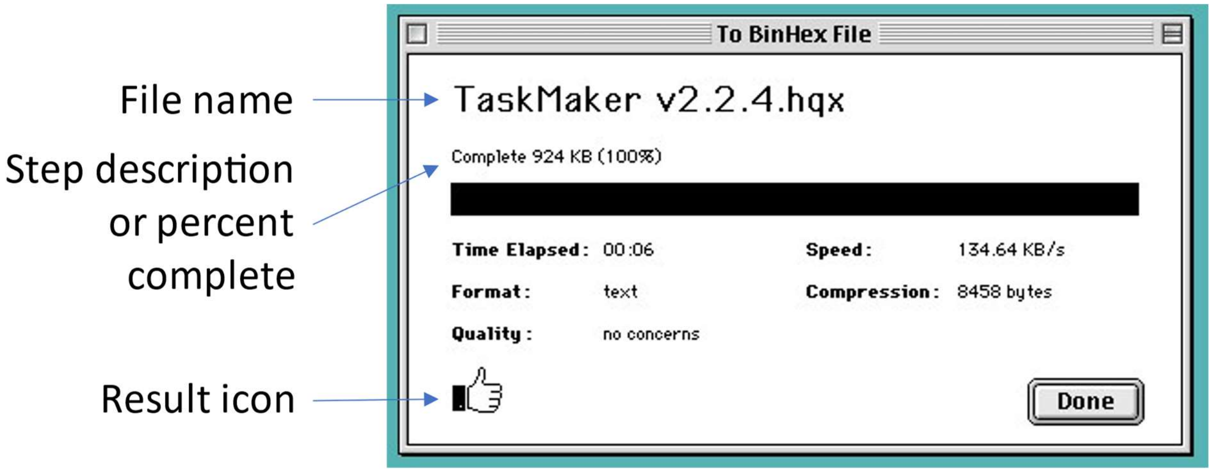
If a receiver computer is connected via a serial cable, and the receiver is running Tiny Transfer, then the receiver may set its clock based on the sender’s date and time. This is automatic, and there is no indication to the user when this happens.

Here are the conditions that determine if the receiver accepts the sender’s date and time:

- The sender’s date and time do not appear corrupted (sanity check)
- The sender’s year is at least 2024
- The receiver is not an emulator. It is assumed that emulators already have the correct clock.
- The receiver’s year is either before 2024 (dead battery) or the sender is an emulator (more accurate clock).

Tiny Transfer Progress Window

During and after operation, Tiny Transfer displays a progress window.



To BinHex File

TaskMaker v2.2.4.hqx

Complete 924 KB (100%)

Time Elapsed: 00:06 Speed: 134.64 KB/s

Format: text Compression: 8458 bytes

Quality: no concerns

Percent complete is measured in actual file size, not the size of the BinHex text, regardless of whether you are converting to/from or sending/receiving.

Time Elapsed may be slightly different for the sender vs the receiver, as the receiver may still have some processing to do even after the sender is finished transmitting.

Speed is actual file size (not BinHex text size) divided by time elapsed. Because the internal timer has additional decimal points beyond the seconds displayed on screen, the speed value may not match a manual calculation based on screen integers.

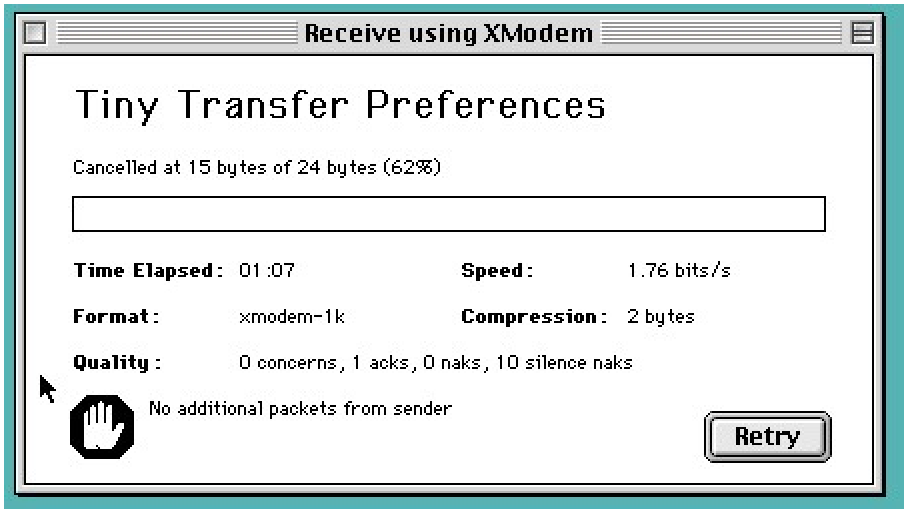
Format is 'text' unless using XModem. For XModem, the specific variety is displayed. A character in parentheses is displayed for XModem varieties other than 1K to explain why the variety was chosen. This is purely informational doesn't represent a problem.

(F)	The file is small. XModem-CRC is more efficient for small files.
(B)	The baud rate is slow. More frequent updates can be performed with 128 byte packets
(N)	The receiver declined XModem-CRC or 1K
(S)	The receiver did not respond to XModem-CRC or 1K
(1)	The receiver did not respond to XModem-1K
(K)	The receiver declined XModem-1K
(U)	You chose something other than XModem-1K in File->Preferences

Compression is the number of bytes saved by the BinHex 4 RLE90 compression algorithm. A positive number is good, and negative number is not. Because the algorithm is required by the BinHex 4 standard, there is no way to disable it when it produces negative results.

Upon successful completion, the progress window's button is labeled 'Done' and the percent message starts with the word 'Complete'. If Tiny Transfer encounters recoverable issues, like extraneous characters or Xmodem retries, there may be messages displayed under **Quality** and a warning icon may appear. However, as long as the button is labeled 'Done', the operation was successful.

Upon cancelation or failure, the button will be labeled ‘Retry’ and issues will be noted under **Quality**. In the following example, someone pulled the serial cable during operation. Under **Quality**, the receiver expresses 1 successfully received packet (acks = acknowledgements) and 10 failed attempts at trying to get the sender to start sending again (naks = negative acknowledgements, silence naks = it was quiet so I sent a negative acknowledgement to try to continue the conversation).



Tiny Transfer Preferences

Cancelled at 15 bytes of 24 bytes (62%)

Time Elapsed: 01:07 **Speed:** 1.76 bits/s

Format: xmodem-1k **Compression:** 2 bytes

Quality: 0 concerns, 1 acks, 0 naks, 10 silence naks

No additional packets from sender

To reiterate, Tiny Transfer can handle some less-than-ideal situations and successfully complete. Because of checksums, the file integrity is assured. But, if there is an unrecoverable failure or too many retries, Tiny Transfer will delete the partially completed file and alert that success was not possible.

No Need to Close the Progress Window

Regardless of whether the operation was successful or not, you do not need to click the button or the close box. The old window will go away when the next operation begins.

Limitations

Tiny Transfer has been tested at all baud speeds from the slowest to the fastest classic Macintosh computers and usually works without issue. However, there are a few limitations depending on the computer or operating system version. These limitations usually occur on floppy-only systems and systems prior to System 7.

Mac 128K

Tiny Transfer is at the limit of the original Mac's memory space. If you use a larger system (such as adding a debugger, init, or disk cache), you may encounter a bomb box with ID=25. This indicates the Mac has run out of memory. The solution is to use a bare-bones system on the Mac 128K.

Finder Comments in Pre-System 7 MultiFinder/Switcher

Tiny Transfer can read and write file comments for all systems. However, this optional step will be skipped if the Finder blocks access to the disk's comment storage. This may happen if the Finder is running at the same time as Tiny Transfer in Mac OSs prior to System 7. If you see a warning in Tiny Transfer, and comments are important to you, the solution is to temporarily turn off MultiFinder (Special menu->Set Startup) or Switcher (remove the system extension) in those older systems.

This inability to read/write file comments with the Finder running is not an issue with Tiny Transfer. It is a restriction of the older operating systems in sharing access to the comment storage file.

Downloading to a Floppy with a Disk Cache on a Slow Mac

Tiny Transfer can convert to/from BinHex format on floppy drives without issue. However, downloading over serial to a floppy can fail due to timeouts, because the sender is not hearing back from the receiver in a timely manner. The cause is often the Macintosh OS disk cache, set in the Memory control panel (RAM cache in the General control panel in System 6 and earlier)

Initially, the Macintosh OS stores the incoming file data in memory (the cache). When the cache fills up, it starts writing to the disk. It appears that the OS decides to write a large chunk at a time. When this happens on a slower computer (like a Macintosh LC without a hard drive), execution of Tiny Transfer essentially halts for a while as the CPU concentrates on the floppy drive. Because the OS is writing a big chunk all at once, the serial port is not receiving enough attention. Interestingly, even the computer's internal timer (TickCount) starts missing counts. It just seems like some computers do not have enough bandwidth to keep up.

You can work around this by:

1. Using a faster computer to receive
2. Using a hard disk or RAM disk and then copying the file to a floppy if desired
3. Or, reducing the size of the disk cache to 32 KB or less

Downloading During AppleTalk Activity

Apple acknowledges that AppleTalk activity interferes with the modem serial port (see Develop issue 9, pages 86-87, Winter 1992). It also cites floppy activity, virtual memory, and even heavy video. The XModem protocol should be able to retry packets if short periods of corruption occur. However, if AppleTalk monopolizes the processor for an extended period, then eventually even XModem will give up.

If you have AppleTalk running and it is causing conflicts:

1. Disable AppleTalk
2. Or, use AppleShare to transfer files.

Encoded File Format

Tiny Transfer represents all files in BinHex 4 format. Tiny Transfer can convert any file type to/from BinHex 4 format. Tiny Transfer cannot convert to/from MacBinary, StuffIt .sit, earlier/later BinHex (i.e. BinHex 2 or BinHex 5), or other formats.

This is no restriction on the type of file Tiny Transfer can send, regardless of the recipient. If communicating with a remote copy of Tiny Transfer, there is no restriction on the type of file Tiny Transfer can receive. This is because Tiny Transfer wraps the file in a BinHex format prior to sending.

However, if a terminal application is sending a file to Tiny Transfer, the file must be in BinHex format.

Why BinHex 4 Format?

BinHex 4 is a popular format for archiving classic Macintosh files. The file names usually end in ".hqx", although ".txt" is also common.

- BinHex 4 preserves the data fork, resource fork, and owner/creator information
- BinHex 4 format is text based
    - It can be recognized by opening the file in a text editor and seeing the "(This file must be converted with BinHex 4.0)"
    - The user can even copy and paste the contents into an email or online posting
- BinHex 4 is an open, documented format. Various repositories even contain algorithm code
- Several existing popular programs, such as "StuffIt" and literally "BinHex 4", support this format

Differences From BinHex 4.0

The following are the contents of the Wikipedia example BinHex file when encoded by Tiny Transfer.

```
(This file must be converted with BinHex 4.0)

:$f*TEQKPH#jdCA0d,R0TG!"6594%8dP8)3%!!!!"I!#3"!&j8dP8)3!"!!!"I(*

-BA8#r`!!!"Err`d0#f*TEQKPH#jdCA0d!*jLJ!!9k`S!!!#!!*!&TSDZIkJ!N"V

rN!4849K89%*#0J%!V-H8mUc(P2)!!!'%!!!!33!!!,d!!!!jeZ1,5!#3"Vha%3!

)(231",%&S!2,Qd`(Cq)DZ!hRBBpMDNcU%8a1$J9f!+AbpSZ`0$V%*fQQe&iYU@F

Se+T9I8H+[V6$HNH[h[HPE)9X2mZ6kH"UPN[hAPLVp[c1%D8c2V2GZcdbqJ(NJj2

1kAqRaKqd(Ad50P2(ha66QC'mQ6i[j)Q0FEMRR9Jh9!,N`HL0pbrPI3Idh'53!+0

34q-bjG"d4mX2GZi@F(MKB*%L(Dl%2*fp(NNVqm"-,Gr42M96mm&apjDb!&%'[aG

N3`iJRcXBc!DqIUc2Jd%BC[N`i`pP!LlXbq##`"m#ET!!i3rk!)Pa%-iLY,H!9Li

4lJF!!!"SH!#3"!!!:


(One more thing...)

:-3C8G'Cb!3"6!JJ36!3!`!)"3a-a16!d,6!a,6!a9$!`1M!`1M!`64-a16!d,6!

a,6!a9$!`1M!`1M!`3K-a16!d,6!a,6!a9$!`1M!`1M!`93B!N!CA!3"B!3""$e0

dG@CQ5A3J4'9XGAKPUPS#)SF!:
```

Both files have the same introductory line "(This file must…", same character set, same line length, and the data begins and ends with a colon character. The data contains the same header format, CRC 16 checks, and so on.

The Tiny Transfer differences are:

1. Tiny Transfer adds a footer that starts with "(One more thing…)". This contains the file dates, comments, and other file information. In the unlikely event a decoder application is not compatible with the footer, the user can always delete the footer using a text editor.
2. Tiny Transfer pads the end of the each data section with ! to make it an even group of four characters. This matches base-64 character set best practices for consistency, security, and concatenation. (https://en.wikipedia.org/wiki/Base64#Output_padding)
3. Tiny Transfer does not clear the invisible and other Finder flags during encoding, only during decoding, per Peter Lewis's BinHex definition. This preserves the original file values. However, it also causes a difference in compression in that area, which causes a difference in data length and encoded characters thereafter.
4. Tiny Transfer does not compress sequences of only three characters, as it literally offers no savings (the RLE90 compression sequence is also three bytes) and only increases entropy.
5. Tiny Transfer compresses sequences of 0x90, just like the original BinHex application. In contrast, StuffIt does not follow this rule, which can result in larger files by that application.

These encoding differences are compatible. Both the BinHex 4 application and StuffIt are capable of decoding Tiny Transfer files perfectly without complaint. Of course, they ignore the footer with the extended file information.

DeHQX v2.0.x and HQXer v1.1.x also decode the files perfectly, but complain about the footer. In DeHQX, you need to click the 'Just Continue' button in the alert. There is no reason to use these applications now that Tiny Transfer is available.

Comparison with BinHex 4.0 Application

There are at least two versions of the BinHex 4.0 application written by Yves Lempereur circa February, 1985. This application is small, around 6 KB in size. It is beautiful and elegant in doing exactly what it was designed to do. It is the basis for the BinHex 4 format. As a testament to the prowess of its author, it seems to be compatible with all classic Macs and Mac OSs, despite being written in early days.

Because it is so small, the BinHex 4.0 application is a good choice for including in ROM Disks or anywhere disk space is at a premium.

In comparison, Tiny Transfer is faster, has a progress GUI, can transmit/receive files, and includes additional file information (comments, dates, and so on). But, this comes at the cost of using more disk space (60KB).

Comparison with StuffIt Lite 3.6

StuffIt compresses and decompress a variety of file formats, including BinHex 4. StuffIt can combine multiple files in a single archive and provides better compression if the .sit format is used first. StuffIt produces worse compression if BinHex 4 format is used alone.

In comparison, Tiny Transfer is faster, can transmit/receive files, has a smaller memory/disk footprint, and is compatible with even the oldest Macintosh computers and operating systems.

The interesting philosophical question is, “In modern times, is it better to archive files in a smaller proprietary format, or in larger open format?” Often, the choice is dictated based on whether a group of files need to be archived together. Amusingly, many Mac archives have a .sit file encoded in BinHex compressed in a .zip.

Comparison with Floppy Emu and SCSI Emulators

For bulk copying, hardware-emulating mass-storage is far faster and convenient. Just drag the files over using the Finder.

Tiny Transfer is useful when you need to send an individual file back and forth, such as during software development. Or, when you only need to copy a few files back to your main Macintosh, such as screen shots or Norton test results. Tiny Transfer also encodes the file such that it can be posted to a forum or online repository.

Another possible reason to use Tiny Transfer rather than a device emulator is if the SCSI ports or floppy drives are in disrepair or missing.

Comparison with Terminal Applications

Macintosh terminal applications usually transmit Macintosh files in MacBinary format by default. Because MacBinary doesn't encode bytes to the ASCII character set, these files are 25% smaller than BinHex 4. Additionally, the MacBinary header captures file information beyond what is included in classic BinHex 4.

Due to lack of compression, MacBinary compares unfavorably to .sit. Due to the binary format, MacBinary compares unfavorably to BinHex. Thus, over the years, users migrated one direction or the other away from MacBinary. Tiny Transfer does not recognize MacBinary. Therefore, the terminal application should be configured to not use that format.

Terminal applications offer a wider variety of transmission options, versus Tiny Transfer. However, the versatility also makes them more complex to set up and use, particularly given that they usually expect modems between the computers.

Terminal programs have mixed results on performance. If they are displaying incoming content on the screen, such as during plain-text transfer, these programs tend to corrupt received files unless using very slow baud rates. However, if the terminal application has implemented XModem, then it can work at higher baud rates. The time it takes for them to calculate the CRC and return an ACK can cause slower performance.

Besides the inability to convert local files to/from BinHex format, the biggest issue with terminal programs tends to be their memory/disk footprint and OS support. The older terminal programs have small footprints but sometimes non-standard implementations of XModem. The new terminal programs tend to have nicer GUIs and decent XModem support, but do not run on the oldest computers.

Speed Tests

Surprisingly, even the original Macintosh with System 1.0 (aka 0.97) is capable of communicating at the maximum 56,700 baud rate!

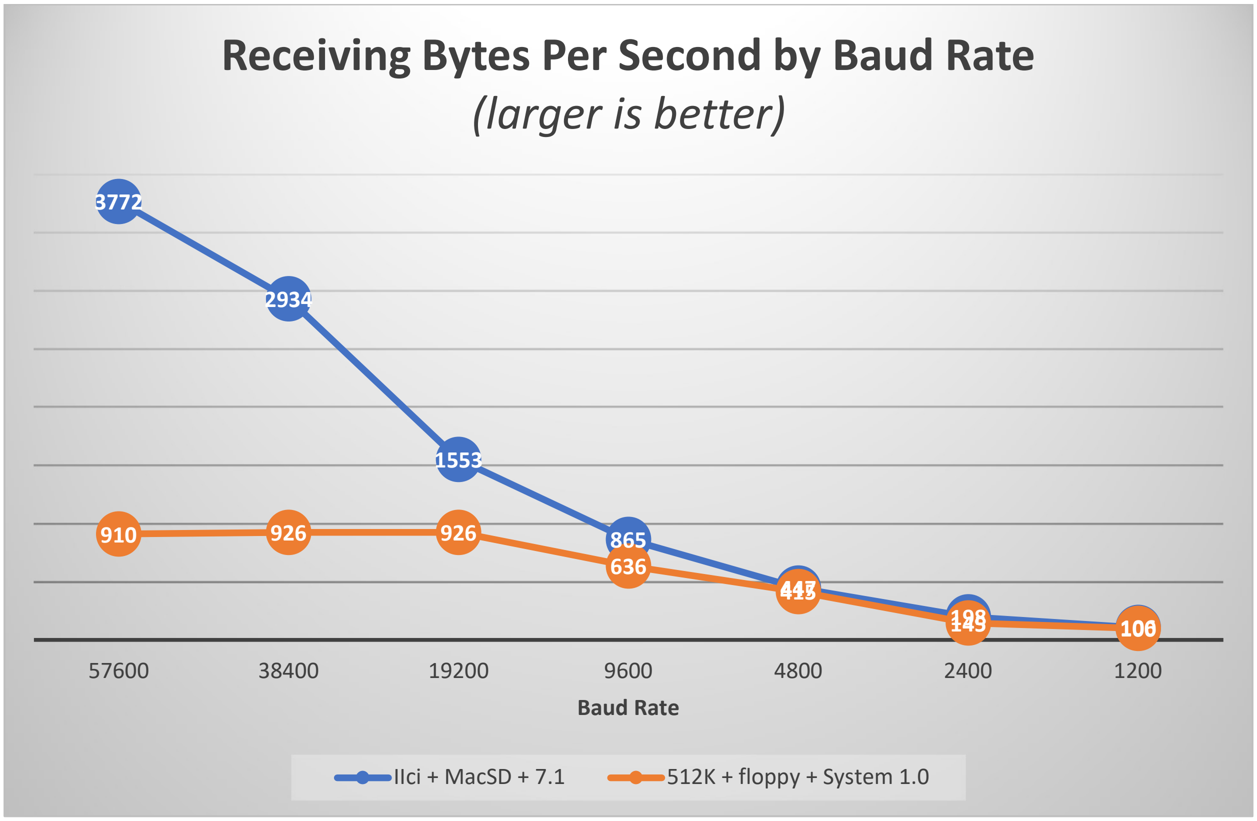
It is more difficult for a slow Mac to receive than it is to send. That's because the slow Mac must copy the incoming bytes before they get overwritten by new arrivals. In contrast, when sending, the slow Mac just has a little more "quiet space" between bytes or packets. The fast receiver doesn't care.

Although the slowest Mac can keep up with the incoming bytes, it doesn't get as much spare time to decode, decompress, write files, and update the GUI. Therefore, it is a little slower to validate the incoming packet and send back an acknowledgement to the fast sender. Those pauses add up, such that a file being sent to a slow Mac at 57,600 takes appreciably longer than sending to a fast Mac.

Occasional retries and repeated packets (the sender gets impatient) are not unusual and do not impact the resulting file. That being said, some terminal applications don't implement XModem correctly (MacTerminal 1.0) or have such short timeouts that they can never receive a packet at slow baud rates before complaining. This is an issue with the terminal application.

CPU & Floppy Bottleneck at Maximum Baud

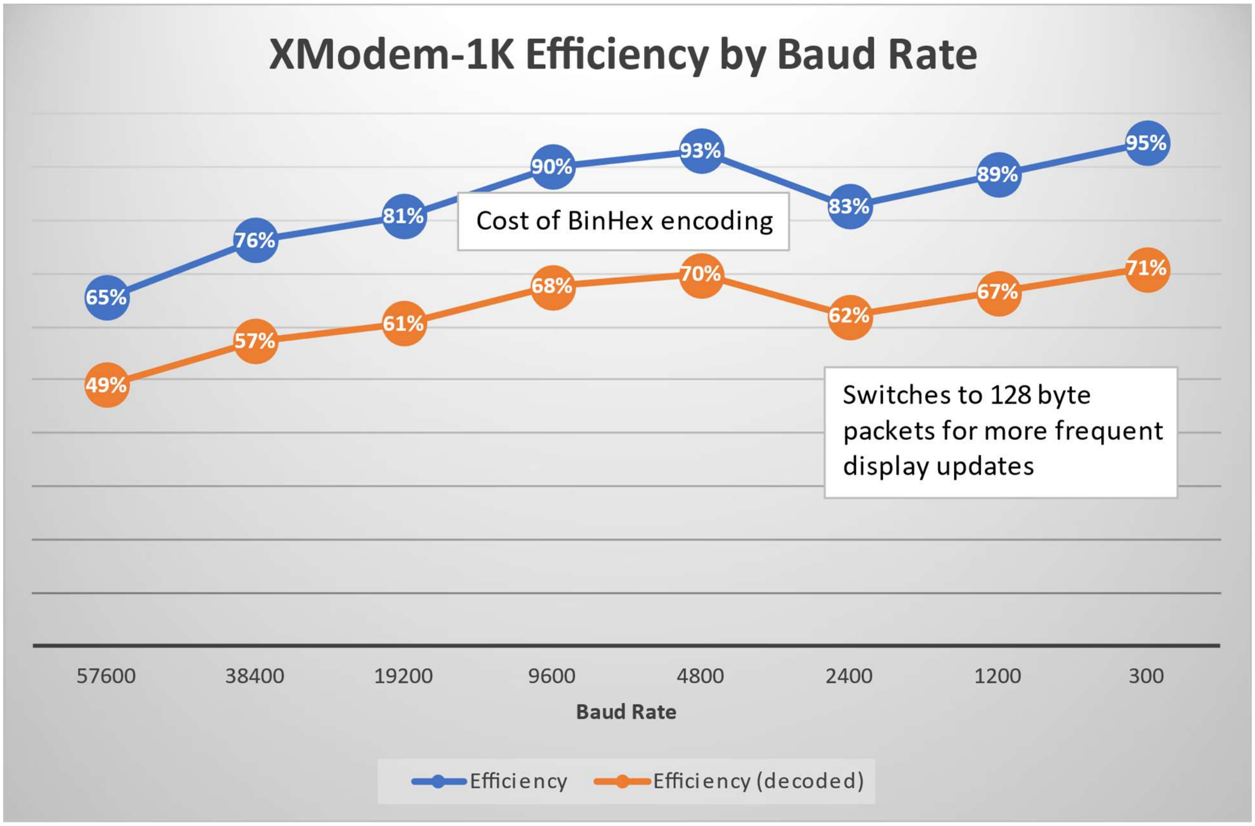
The following chart shows that a slow Mac (orange line "512K") still works at the fastest baud rate on an error-free serial connection. But, starting around 19200 baud, the CPU and floppy become the bandwidth limit. In comparison, a Macintosh IIci (blue line) continues to benefit from faster baud rates.



Efficiency of Serial Transfer

Even the best computers can't actually reach full 57,600 bit/s *throughput*, although they work at that baud rate. There is transmission overhead consisting of start bits, stop bits, start bytes, headers, checksums, BinHex character encoding, and XModem response latency. The last two are the most significant.

BinHex 4 uses ASCII characters to represent bytes. This results in a 4/3 increase in size, which reduces transmission efficiency. In the graph below, the gap between the top (blue) and bottom (orange) lines shows the cost of encoding the transfer rather than using straight binary.



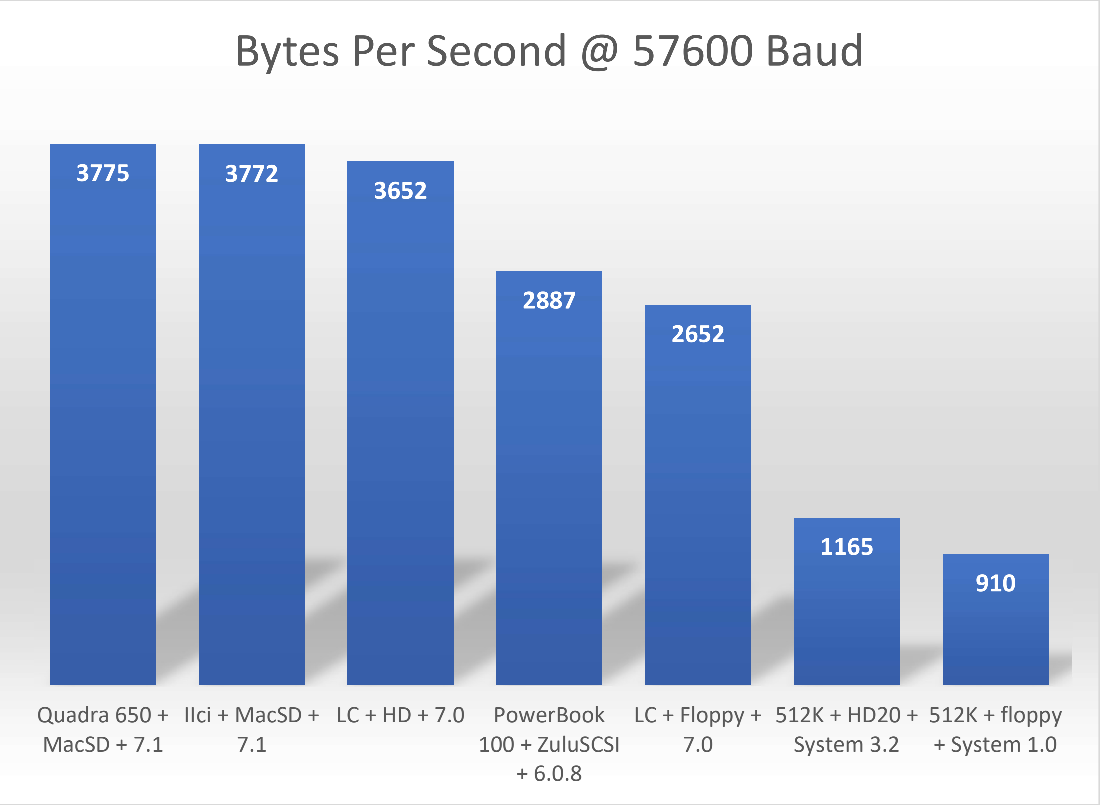
XModem waits for a reply for each packet, which idles the serial port. In the graph above, the slope of the bottom line (orange) demonstrates how the cost of waiting for a reply hits the fastest baud rates the most.

Unseen is the relatively poor performance of RLE90 compression. When run on The Calgary Corpus (excluding the pic file), it only saves 1%. Shockingly, the total result jumps to 14% when including the pic file. This behavior is consistent where most of the time the compression is unremarkable, but occasionally it saves significant space on disk images and other files with lots of repeating characters.

Tiny Transfer could increase serial performance by using a modern set of encoding, transfer protocol, and compression methods. However, Tiny Transfer was designed to prioritize compatibility with older terminal programs and file formats.

Macs that Maximize Serial Transfer

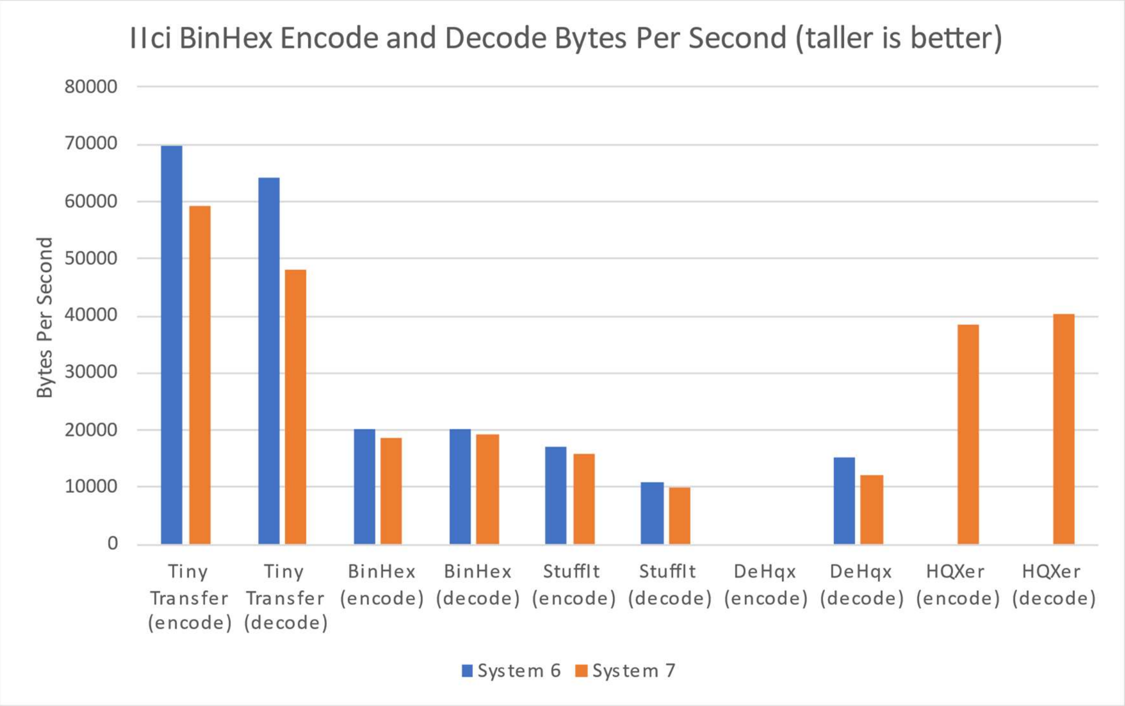
Floppy-based computers or those running original 68000 CPUs take longer to acknowledge receiving a packet. As described earlier, the time spent waiting could have been used to transmit additional data. The delay is virtually eliminated for a Macintosh running a 68020 or better on a hard drive. Even faster computers make little difference.



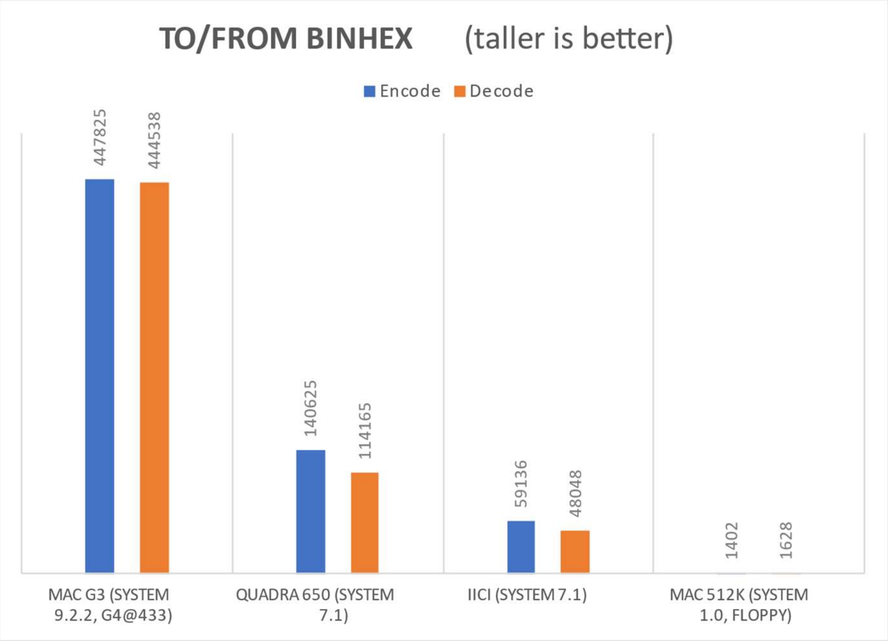
Speed Comparison of File Encoding and Decoding Applications

Using a Macintosh IIci without a cache card, various BinHex applications were timed to encode and decode 'book1' of the Calgary Corpus. These tests were first performed in System 6.0.8 without MultiFinder, and System 7.1 with integrated MultiFinder.

Tiny Transfer is significantly faster than the BinHex application, StuffIt, and DeHqx.



Using just Tiny Transfer, various eras of Macs show the performance gains Apple achieved.



Technical Information

The remainder of this document is for software developers.

BinHex 4+1 Format

As described earlier in this document, Tiny Transfer extends the BinHex 4 format by appending a footer named “One more thing” after the usual BinHex data block.

```
(This file must be converted with BinHex 4.0)

:blahblahblahblah:

(One more thing…)

:blahblahblah:
```

This should be compatible as BinHex files often have text before and after them because they were captured from terminal logs or sent as part of an email. Thus, the most popular decoder applications ignore everything outside of the primary BinHex block. In the unlikely event you encounter a completely incompatible decoder, you can just manually delete the footer. The beauty of BinHex is that it is a text file. The footer contains non-critical file properties and the decoded file can still be generated, just without the extra file info.

The footer uses the same encoding as the primary section. A program can process it as follows:

1. After the final colon of the primary BinHex section, look for the partial string “(One more thing” at the start of a line.
2. Decode the characters between the colons, using the standard BinHex 4 character set
3. Decompress the bytes using RLE90, as per BinHex 4

That leaves you with file info data in a binary state. It follows a repeating pattern:

<type><length><content>(repeating)

Where:

<type> is a single alphanumeric character (0…9, A...Z, a…z)

<length> is a single byte (0…255) describing the length of the content that follows. This does not count the single byte of the type or length itself. It only counts the content that follows.

<content> 0 to 255 bytes

The footer always ends in this final sequence:

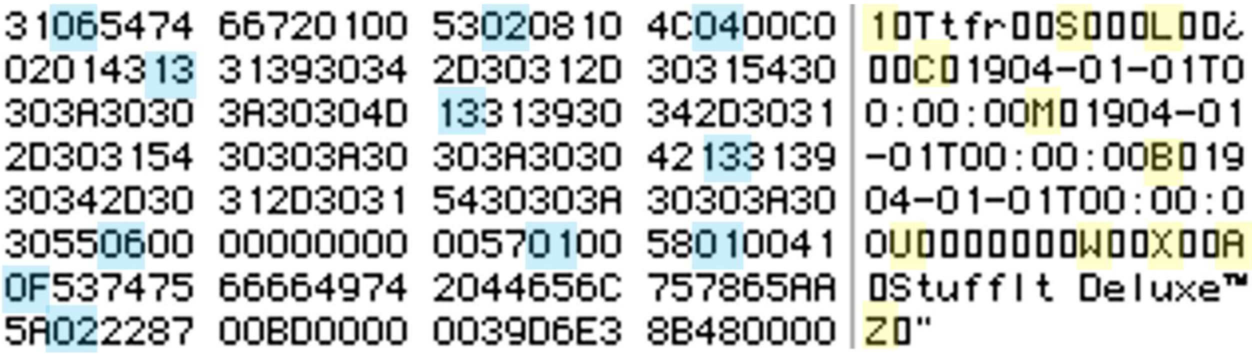
Z<0x02><two bytes: CRC16>

Calculate the CRC from the start of the decoded, decompressed footer data all the way through to the end of the CRC (but act like the two CRC bytes equal <0x0000>). This is the same algorithm as classic BinHex data. After the checksum pattern, ignore any remaining bytes, which is usually just zero padding.

If any type isn’t alphanumeric, the CRC is missing, the CRC doesn’t match, or the footer is missing, then the footer is considered bad and should be abandoned. You should still decode the primary BinHex file, as the footer is considered non-critical.

Example of Footer Contents

Putting them all together, here is an example “one more thing” footer after decoding and decompressing:



```
31065474 66720100 53020810 4C0400C0 | 1□Ttfr□□S□□□L□□¿
02014313 31393034 2D30312D 30315430 | □□C□1904-01-01T0
303A3030 3A30304D 13313930 342D3031 | 0:00:00M□1904-01
2D303154 30303A30 303A3030 42133139 | -01T00:00:00B□19
30342D30 312D3031 5430303A 30303A30 | 04-01-01T00:00:0
30550600 00000000 00570100 58010041 | 0U□□□□□□□W□□X□□A
0F537475 66664974 2044656C 757865AA | □StuffIt Deluxe™
5A022287 00BD0000 0039D6E3 8B480000 | Z□"
```

The yellow highlights on the left are the types (described in the next section). The blue highlights on the right are the lengths. Notice some readable text like StuffIt Deluxe (the creator of the file inside the BinHex text) and some dates starting with 1904. The source file came from Wikipedia and demonstrates the problem with the standard BinHex 4 format. That format does not include dates, so those dates default to the start of the Macintosh clock. If the original file had been converted using Tiny Transfer, the actual original file dates would have been preserved.

Footer Types

Here are the currently supported types:

Encoder Application Type

```
        #define kOneMoreThing_StartType          '1'
```

At the beginning by convention. This relates to the application that encoded the BinHex file. The application creator 'Ttfr' (use your own if you write this footer) and application version in most-significant-byte-first order and binary coded decimal. So, 1.52 is <0x01><0x52>. Written but currently ignored on read.

File Application Type

```
        #define kOneMoreThing_ApplicationType    'A'
```

For documents, a human-readable pascal string with the name of the application that created the original document (such as 'MacPaint'). This differs from the application that encoded the document into BinHex. See FileCommentType for string format information. Written but currently ignored on read.

Date Types

```
        #define kOneMoreThing_BackupDateType    'B'

        #define kOneMoreThing_CreationDateType  'C'

        #define kOneMoreThing_ModifiedDateType  'M'
```

In ASCII ISO 8601 YYYY-MM-DDTHH:MM:SS format. Optional 'Z' or timezone offset at end. Otherwise local time. Note that this is a 2040 friendly format. If the Mac OS gets patched to recognize years beyond 2040, then Tiny Transfer will store/restore correct years. The StuffIt .sit format uses a 32-bit value of seconds since 1904, and thus cannot explicitly store correct dates after 2040.

Tiny Transfer currently ignores timezone offsets.

Directory Type

```
        #define kOneMoreThing_DirectoryType     'D'
```

Where the file should reside in a multifile archive. ':folder:' or ':folder:subfolder:'. Always include colons. Currently not implemented.

File Comment Type

```
        #define kOneMoreThing_FileCommentType   'F'
```

Pascal string for the Get Info comment. Use the content length for the start of the Pascal string. For example, in <F><0x07><Comment>, the start of the Pascal string is the <0x07>. The string does not end in a <0x00> so don't use a C string copy routine.

Folder Location Type

```
        #define kOneMoreThing_LocationType      'L'
```

Two shorts that represent the graphic point to display the file in the folder (fdLocation). <vertical><horizontal>. This is a Macintosh Point. This information is written by Tiny Transfer, but currently not restored, as it seemed to result in a messy destination folder.

System Version Type

```
        #define kOneMoreThing_SystemVersionType 'S'
```

System Version In 2 byte MSB BCD, such as <0x07><0x01>. Written but currently ignored on read. Note that the first Macintosh System is actually System 0.97 <0x0097>, although by convention we refer to it as "System 1.0".

Finder Unused Bytes Type

```
        #define kOneMoreThing_UnusedFlagsType   'U'
```

The three unused Finder shorts fdUnused[3]. Ignored on read because these are undocumented. It might produce bad results if restored to a different disk or system.

File Name Script Type

```
        #define kOneMoreThing_ScriptType        'W'
```

File name script language fdScript.

Finder Extended Flags Type

```
        #define kOneMoreThing_XFlagsType        'X'
```

Extended finder flags fdXFlags. Currently ignored on read because these are undocumented. It might produce bad results if restored to a different disk or system.

CRC Footer Check Type

```
        #define kOneMoreThing_CRCType           'Z'
```

CRC16. Always required at the end. We stop processing here.

Extensible

The footer section is extensible. That is, you can add your own types. Use a lowercase letter (uppercase and numbers are reserved by Tiny Transfer). Always end with the ‘Z’ type (CRC16).

The great thing about this pattern is that a decoding program doesn’t need to know or care about every type. Just read the type character, read the length, if you don’t care about this type, skip that many length bytes to reach the next type. When you reach a ‘Z’, you’ve reached the checksum at the end.

Other than the checksum, the types can appear in any order or may not appear at all. For example, an encoder might just choose to encode the creation data, backup date, and CRC. The Tiny Transfer reader consists of a while loop (remaining bytes) with a case statement on the type. It works regardless of order, repeats, missing types, new types, etc.

Technical Information about Finder Comments

Apple has always hated file comments. Okay, that's an exaggeration. But they've been treated as disposable second-class citizens. The root cause is that Finder comments are not stored in the file itself or in the disk file structure. Instead, file comments are owned and managed by the Finder application, in a special invisible file. That was a bad design choice.

Prior to the introduction of the Desktop Manager (circa System 7), the Finder stored all comments for the files on a disk in a hidden resource file named 'DeskTop'. The comment resource type is 'FCMT'. Three numbering schemes were employed:

System 1.0: Random resource ID. The file ID (ioFlNum) is stored in the first four bytes of the 'FCMT', followed by a Pascal string. This method requires the Finder to read through all of the comments until it finds the matching file ID.

System 1.1 until HFS: 16-bit hash resource ID (see Macintosh Technical note #29). 'FCMT' is just a Pascal string. No leading ID bytes. Fast lookup by just asking for the FCMT resource with the hash ID. However, the hash could result in two files having the same value (a hash collision) and sharing the same comment.

This algorithm and resource structure is not compatible with System 1.0. Thus, Finder 1.0 cannot see Finder 1.1 comments, nor vice versa. In fact, hilariously, you can get the same file to have different comments in System 1.0 versus System 1.1 and up.

HFS through System 6.x: Random resource ID which is then written to the drive structure under the file property `paramBlock.hFileInfo.ioFlXFndrInfo.fdComment`. 'FCMT' is just a Pascal string. No leading ID bytes. Fast lookup by just asking for the FCMT resource with the fdComment ID. No more collisions since each file is given a unique id.

This algorithm is not compatible with earlier systems. However, this algorithm is only used on HFS formatted disks. MFS disks continue to use the System 1.1 algorithm. So, no problem.

Finder Monopolization of the DeskTop File

Besides the variety of algorithms and structures for storing file comments, the other problem is that the Finder opens the DeskTop file and keeps it open. This can deny access by other applications, like backup utilities and Tiny Transfer.

In early systems, this wasn't a problem because only one program was being run at a time. The Finder would close the DeskTop file, end operation, and launch the other application. However, with the advent of Switcher and MultiFinder, the Finder would keep running alongside the other applications, keeping exclusive access to the DeskTop file containing all the comments. Additionally, if a user recently modified a comment, the Finder may not yet have triggered the resource write to the disk.

The user needs to disable MultiFinder/Switcher to guarantee other applications access to the DeskTop file and the recently updated comments.

Enter the Desktop Manager

The Desktop Manager is a set of operating system routines that take over some of the Finder's previous responsibilities. The Desktop Manager is built into System 7 and beyond, and is available as an INIT for earlier systems, such as System 6. The Desktop Manager is a huge step forward, as multiple applications can read comments, determine file owners, and even obtain file icons.

There is one problem: This only applies to disks that the Desktop Manager decides to manage. For example, MFS floppies and HFS floppies aren't managed. This is probably because floppies commonly moved between computers. Retaining the original classic "DeskTop" file avoids having the "Desktop rebuilt" for the floppy when inserted into computers with different OS versions.

For programmers, an application must first call `PBHGetVolParms()` with the requested disk volume number and then check the `vMAttrib` flags for the `bHasDesktopMgr` bit. If it is set, call the Desktop Manager. If not, then open the resource fork of the DeskTop file the old-fashioned way.