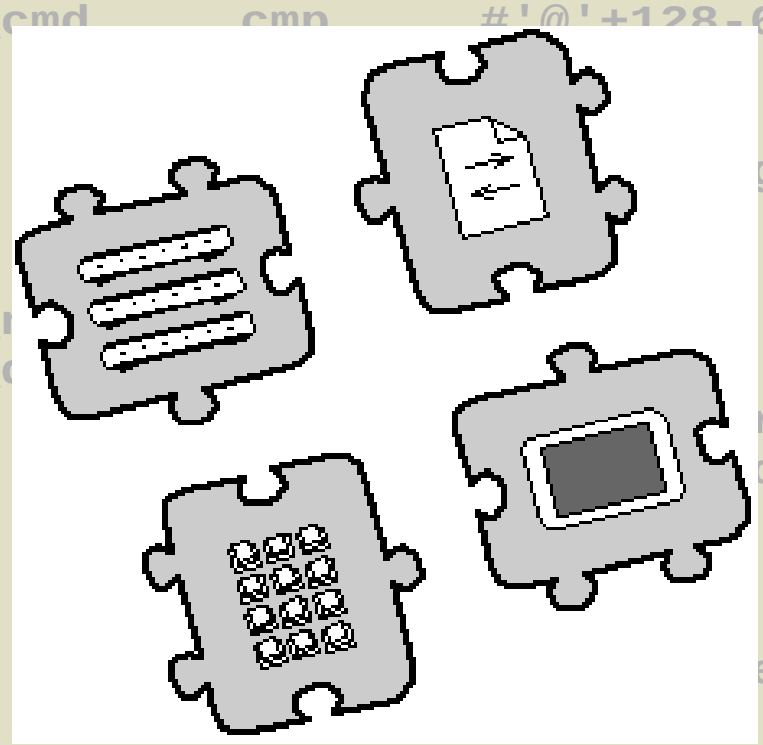
ModemWorks

Technical Reference

```
                sta     prmtbl
                ldy     #SerWriteChar
                jsr     PortTool
                bra     term_read

term_cmd        cmp     #'@'+128-64


                                gMode


term_r
term_c                                  rsor
                                        ol


                                        e


updateTerm      ldy     #SerReadChar
                jsr     PortTool
                ldy     updateCycle
```



Morgan Davis Group

Copyright

HTTP://WWW.MORGANDAVIS.NET

FIRST PRINTING — MAY 1992 — U.S.A.

Printed on recycled paper.

Contents

Getting Started

ModemWorks lets you develop high performance data communications software in BASIC as well as in assembly language. Although ModemWorks comes with everything needed to create communications programs on your computer, its modular design offers "plug and play" expandability, allowing you to add additional features. This **ModemWorks Technical Reference** shows you how to access ModemWorks' modules from assembly language programs. It also describes the interfaces to the various modules so that you can integrate new modules with those that already exist.

This chapter introduces you to ModemWorks and its technical origin. It begins by describing the things you need to know before developing your own modules.

What You Should Know

Before embarking on the process of creating a custom ModemWorks module, you should possess the following:

- An assembly language development system
- Knowledge of 65C02 programming
- An understanding of basic data communications concepts
- The Object Module Manager (OMM) and manual
- Familiarity with the OMM, module format, IMC, etc.

This manual assumes that you possess these qualifications. As the title of this manual suggests, the information presented is quite technical. It is not for the casual programmer.

Required reading: Chapter 3, **ModemWorks Modules**, in the **ModemWorks BASIC Communications Toolbox** manual.

Past, Present, Future

Historically, ModemWorks did not adopt an open architecture until the 3.0 version was released in 1992. In this industry, things change at breakneck speed. What may push the limits of technology today will amuse us by its prehistoric nature tomorrow. ModemWorks may have been on the cutting edge in 1984, when 1200 bps modems were amazing, but it was doomed by a closed architecture. The modem industry soon outpaced computers and software, even ModemWorks.

Unless your Apple IIGS is equipped with an accellerator, current technology already exceeds the ability of the computer to keep up with high-speed communications. If the hardware can't handle it, software doesn't stand a chance.

In 1990, work began on rewriting ModemWorks from scratch. It was to be based on an open architecture. The major parts of ModemWorks would be serviced by interchangeable modules. To software that called upon these modules, they would all seem to operate identically, even though they may integrate with a myriad of devices. With a standard interface accessible to software, programs could finally get work done without being concerned with hardware peculiarities.

During the development process, the Object Module Manager was born. The OMM is the heart of ModemWorks. It allows ModemWorks' modules to communicate among each other, sending commands and making requests. It makes the integrated, open structure successful.

Let us not believe that even with this new architecture that we will enjoy the cutting edge forever. We may not be communicating using modems in the next eight years. Undoubtedly, new technology will make modems obsolete. Perhaps we'll connect via high-speed links over direct connections handled for us by the phone (or cable TV?) company. No more Hayes-style AT commands. No more voice-grade lines. We'll all just "network" with each other like so many computerized television sets.

See you on channel 6502.

Interfaces

This chapter presents the interface to each kind of module that comprises ModemWorks. The interface consists of intermodule-communication command numbers that are passed through the OMM's message passing feature. Each command is explained, including the parameters it accepts as input or output.

Passing & Receiving

Module can exchange information in a variety of ways. Values may be passed in a parameter table. The CPU's registers (A, X, and Y) may be used. Even the processor's flags (zero, carry, overflow) can be used to return information.

In the pages that follow, these symbols describe both input and output parameters:

prmtbl	Memory location $E0
A	A-register (accumulator)
X	X-register
Y	Y-register
C	Carry flag
N	Negative flag
Z	Zero flag
V	Overflow flag

ModemWorks commands use a six-byte area of memory at location $E0 for passing parameters. This location is called **prmtbl**. Ranges of bytes in the parameter table are identified by this notation: **prmtbl[0..3]**. This is shorthand for giving the locations prmtbl, prmtbl+1, prmtbl+2, and prmtbl+3. Another example: **prmtbl[2]**. This denotes the byte at prmtbl+2.

TimeTool

A Time Tool, such as Time and TimeGS, provide a timing system for software. Timing is required by every module in ModemWorks, so this is the most important kind of module in the system. Note that many of these functions require that they be called at least once every 1/60 second in order to provide fairly accurate timing.

```
************************************************
***
***    TimeTool.equ
***

TT_ID          equ    $7474  ;Time Tool ("tt") ID


Ticker         equ    0
```

Call Ticker to find out when the leading edge of the next tick begins. This lets you do your own timing in 1/60 second increments.
Input: None
Output: C=1 if new tick cycle starting

```
GetTicks       equ    1
```

GetTicks calls Ticker for you, incrementing a tick counter. The 16-bit value of the counter is returned in prmtbl[0..1].

```
CountDown      equ    2
```

Use CountDown after setting a count with SetCounter. Repeatedly call it while doing some other task.
Input: None
Output: Z=1 when counter reaches zero

```
WaitTicks      equ    3
```

Call WaitTicks to suspend execution for an interval. If provided, the TimeTool will execute a procedure once every tick cycle. The procedure must preserve all registers. It can force the WaitTicks call to quit early by setting the carry flag before returning. A null procedure argument indicates no procedure.
Input: prmtbl[0..1]=tick count, prmtbl[2..3]=procedure
Output: C=0 when WaitTicks times out.
C=1 when WaitTicks is cancelled.

```
WaitSeconds    equ    4
```

WaitSeconds is identical to WaitTicks, only it suspends execution in one second increments rather than ticks. It also will execute a procedure, if provided, every 1/60 second.
Input: prmtbl[0..1]=tick count, prmtbl[2..3]=procedure
Output: C=0 when WaitSeconds times out.
C=1 when WaitSeconds is cancelled.

```
SetCounter     equ    5
```

Use SetCounter before calling CountDown.
Input: prmtbl[0..1]=tick count
Output: None

```
GetTimeStr     equ    6
```

GetTimeStr returns a descriptor for a 22-character string containing time information. The descriptor is at lowtr ($9B).
Input: None
Output: lowtr[0]=length, lowtr[1..2]=address of string in this format: "Fri, 6 Mar 92 12:54:36"

```
FastCPU         equ     7
SlowCPU         equ     8
```

These functions set the Apple IIGS CPU speed to Fast or Normal (Slow) speed.
Input: None
Output: None

PortTool

A Port Tool is responsible for low-level communications I/O with a serial device.

```
************************************************
***
***     PortTool.equ
***

PT_ID           equ     $7470  ;Port Tool ("pt") ID


SerOpen         equ     0
```

SerOpen opens the serial device specified by its slot number for a communications session.
Input: prmtbl[0]=slot of serial device
Output: None

```
SerClose        equ     1
```

Use SerClose when all operations with the serial device opened with SerOpen are completed. Failure to make this call may leave interrupt servicing enabled and can crash the system.
Input: None
Output: None

```
SerReset        equ     2
```

SerReset reinitializes the serial device previously opened with SerOpen.
Input: None
Output: None

```
SerSendBreak    equ     3
```

SerSendBreak sends a 230ms break signal to the serial device.
Input: None
Output: None

```
SerSetDTR       equ     4
SerClearDTR     equ     5
```

These functions turn on (SerSetDTR) and turn off (SerClearDTR) the Data Terminal Ready signal.
Input: None
Output: None

```
SerSetPortBits  equ     6
```

Use SerSetPortBits to adjust data, stop, and parity bits. Values are:

Data / Stop Bits	Parity Bits
0 = 8 / 1	0 = None
1 = 7 / 1	1 = Odd
2 = 6 / 1	2 = None
3 = 5 / 1	3 = Even
4 = 8 / 2	4 = Mark
5 = 7 / 2	5 = Space
6 = 6 / 2	
7 = 5 / 2	

Input: prmtbl[0]=data/stop bits, prmtbl[1]=parity bits
Output: None

```
SerSetSpeed     equ     7
SerGetSpeed     equ     8
```

These functions set or get the serial port speed. Speed values are:

0 = Default	8 = 1200
1 = 50	9 = 1800
2 = 75	10 = 2400
3 = 110	11 = 3600
4 = 134.5	12 = 4800
5 = 150	13 = 7200
6 = 300	14 = 9600
7 = 600	15 = 19200

Input: prmtbl[0]=speed (for Set)
Output: A-reg=speed (for Get)

```
SerGetDCD       equ     9
```

SerGetDCD returns the status of the Data Carrier Detect signal.
Input: None
Output: C=0 no carrier, C=1 carrier present

```
SerWriteChar    equ     10
```

Writes a character to the serial device.
Input: prmtbl[0]=character
Output: None

```
SerWriteBuffer  equ     11
```

SerWriteBuffer writes from zero to $FFFF characters to the serial device.
Input: prmtbl[0..1]=count, prmtbl[2..3]=data buffer address
Output: None

```
SerReadChar    equ     12
```

Reads a character from the serial device.
Input: None
Output: C=0 no character
C=1 character read, A=character

```
SerReadBuffer  equ     13
```

SerReadBuffer reads zero to $FFFF characters and places them into a buffer at the address specified. Note that SerReadBuffer will not return until the requested character count is met.
Input: prmtbl[0..1]=count, prmtbl[2..3]=data buffer address
Output: None

```
SerFlushInQ    equ     14
```

Flushes any buffered input.
Input: None
Output: None

```
SerGetInQ      equ     15
```

Returns the count of characters in the serial buffer waiting to be read.
Input: None
Output: prmtbl[0..1]=count

```
SerGetInBuf    equ     16
SerSetInBuf    equ     17
```

SerGetInBuf returns the address and size of the serial input buffer. SerSetInBuf instructs the serial tool to use the specified buffer.
I/O: prmtbl[0..3]=input buffer address
prmtbl[4..5]=size of input buffer

```
SerSetFlow      equ     18
```

Adjusts data flow control characteristics for the serial device. Values for the type of flow control are:

0 = Reserved
1 = None
2 = XON / XOFF
3 = RTS/CTS hardware handshaking
4 = RTS input hanshaking
5 = CTS output hanshaking

Input: prmtbl[0]=flow control type
Output: None

```
SerAddCompVec           equ     19
SerDelCompVec           equ     20
SerClearCompVec         equ     21
```

These functions manage the serial input interrupt completion feature. Use SerAddCompVec to assign a completion vector for the procedure address you specify. Use SerDelCompVec to remove completion vectoring for an address. Use SerClearCompVec to remove all interrupt completion vector handlers.
Input: prmtbl[0..1]=address of completion handler
Output: None

```
SerAddSearch    equ     22
SerDelSearch    equ     23
SerClearSearch  equ     26
```

These functions add or remove C-style (null terminated) strings for handshaking by the SerGetSearch or SerShowSearch functions. SerClearSearch removes all strings from the search manager.
Input: prmtbl[0..1]=address of string
Output: None

```
SerGetSearch  equ    24
SerShowSearch equ    25
```

These functions read serial input and matches it against any strings added to the serial search manager by SerAddSearch. Both functions operate similarly, except SerShowSearch sends all characters processed to a Console tool, if available. Searches require repeated calls to these functions as they only read and process one character per call. If a string is found, its address is returned. If no string is found, $0000 is returned.

Input: None

Output: prmtbl[0..1]=address of matched string (or $0000 if none found).

```
SerGetTimedByte       equ    27
```

Use SerGetTimedByte to suspend execution for an interval (in ticks) while waiting for serial input.

Input: prmtbl[0..1]=ticks

Output: C=0, timed out—no input (if V=1, lost carrier)
C=1, A=character read

```
SerOutBuffering       equ    28
```

Use this function to enable or disable serial output buffering (IIGS serial port only). With output buffering enabled, calls to SerWriteChar or SerWriteBuffer return immediately. With output buffering disabled, these calls do not return until the last character is transmitted.

Input: prmtbl[0]=1 enables output buffering
prmtbl[0]=0 disables output buffering (default)

Output: None

```
SerSetDCD       equ     29
```

This function controls Data Carrier Detect spoofing. If enabled, all calls to SerGetDCD return a TRUE status. If disabled, SerGetDCD returns the actual DCD status.
Input: prmtbl[0]=1 enables DCD spoofing
prmtbl[0]=0 disables DCD spoofing (default)
Output: None

ModemTool

A Modem Tool is responsible for interfacing with a modem device.

```
*************************************************************
***
***     ModemTool.equ
***

MT_ID           equ     $746d   ;Modem Tool ('mt') ID


InitModem       equ     0
```

InitModem establishes a new modem session, setting the modem for proper operation with the Modem tool.
Input: None
Output: A=0 if initialization failed
A=1 if initialization was successful

```
ModemExit       equ     1
```

ModemExit terminates a modem session, resetting the modem to its preconfigured settings.
Input: None
Output: None

IsOnline equ 2

IsOnline returns the online status of the modem. This status is regulated by the use of other Modem Tool functions. For example, if HandleConnect is successful, the Modem Tool asserts an online status. Using HangUp disables the online status. If the Modem Tool thinks it is offline, IsOnline returns a zero result. However, if thinks it could be online, it determines the online state by calling SerGetDCD. This handles the situation where carrier is lost during a connection.
Input: None
Output: A=0 offline
A=1 online

HasMNP equ 3

HasMNP returns the modem's error correction capability status.
Input: None
Output: A=0 no error correction ability
A=1 can employ error correction

DialNumber equ 4

DialNumber dials a phone number. If the phone number begins with the letters **AT** the number string is sent directly to the modem. This allows the caller to specify additional modem control commands before dialing. If the number does not begin with **AT**, the Modem tool sends **AT** followed by the modem's commands for adjusting error correction (if available), the commands for dialing with pulses or Touch-Tones™ (as specified), and finally the phone number string of characters.
Input: prmtbl[0]= length of phone number string
prmtbl[1..2]= address of phone number string
prmtbl[3]=Touch-Tones(1) or pulses(0)
Output: None

```
SetBusy         equ     5
```

SetBusy adjusts the off-hook state of the modem.
Input: prmtbl[0]=0 go onhook (not busy)
prmtbl[0]=1 go offhook (busy)
Output: None

```
HandleConnect   equ     6
```

HandleConnect is used after answering or dialing to watch for a connection (or other event, such as a busy signal). It suspends execution for an interval (in seconds) or until the modem returns a connection result. Pressing any key will cancel the attempt.
Connection results are:

0 = connection established
1 = cancelled by a key press
2 = no connection
3 = busy
4 = no dial tone
5 = no answer
6 = voice detected

Input: prmtbl[0..1]=seconds
Output: A=result code

```
AnswerLine      equ     7
OrigAnsLine     equ     11
```

These functions tell the modem to pickup the phone line and send an Answer or Originate carrier tone.
Input: None
Output: None

```
HangUp          equ     8
```

HangUp attempts to terminate the online connection.
Input: None
Output: None

```
IsRinging       equ     9
```

Returns the ringing status of the phone line.
Input: None
Output: C=0 no ring
C=1 ring detected

```
SetMNP          equ     10
```

Enables or disables the modem's error correction feature for subsequent use when dialing or answering.
Input: prmtbl[0]=0 disable error correction (any non-zero value enables error correction)
Output: None

```
ResetModem      equ     12
```

Reinitializes the modem without changing its operating speed.
Input: None
Output: A=0 reset failed
A=1 reset was successful

```
SetSpeaker      equ     13
```

Specifies the modem's speaker mode during connections and online sessions. Values for the mode are:
0 = speaker always off
1 = speaker on until carrier detected
2 = speaker always on
3 = speaker off when carrier detected and while dialing
Input: prmtbl[0]=speaker mode
Output: None

```
GetMode         equ     14
```

Returns the modem's mode. Mode values are:

0 = answer mode
1 = originate mode
2 = quiet mode (offhook, no connection)

Input: None
Output: prmtbl[0]=mode

```
ModemType       equ     15
```

Returns the modem's type. Returned values are:

0 = no modem
1 = internal
2 = external

Input: None
Output: prmtbl[0]=type

```
ConnectSpeed    equ     16
```

Returns the modem's last connection speed. This is the speed at which the modem reported a connection, and is not necessarily the speed between the computer's port and the modem. See SerSetSpeed for a list of speed values.
Input: None
Output: prmtbl[0]=speed

```
SetModem        equ     17
```

Passes the address of a modem capability (modemcap) structure. A modemcap defines various characteristics for the modem.
Input: prmtbl[0..1]=address of modemcap
Output: None

```
SetModemSpeed equ    18
```

Sets the operating speed for the modem. Applications that work with a Modem Tool should use this function rather than going directly to a Port Tool to change the speed. See SerSetSpeed for speed values.
Input: prmtbl[0]=speed
Output: None

ConsoleTool

A Console Tool manages input and output with the console—the keyboard and video screen. It is also responsible for processing terminal emulation requests.

```
*************************************************
***
***    ConsoleTool.equ
***

CT_ID          equ    $7463 ;Console Tool ("ct")ID


CTOpen         equ    0
```

CTOpen opens a session with the video display and keyboard. The caller passes the desired width (in columns) of the display screen (either 40 or 80).
Input: prmtbl[0]=width of display
Output: None

```
CTClose        equ    1
```

Closes a session previously opened with CTOpen.
Input: None
Output: None

```
CTReset         equ     2
```

Resets the console.
Input: None
Output: None

```
CTControl       equ     3
```

CTControl performs a display function such as moving the cursor, turning on inverse video, and clearing sections of the screen. CTControl returns a buffer of control codes to be sent to a remote device for full terminal control of both the local and remote consoles. It is up to the application to send this buffer to a Port tool. Control codes are:

1 = gotoxy	14 = scroll screen up
2 = clear screen	15 = scroll screen down
3 = clear to end of screen	16 = cursor up
4 = clear to end of line	17 = cursor down
5 = insert line	18 = cursor right
6 = delete line	19 = cursor left
7 = insert space at cursor	20 = soft tab
8 = delete char at cursor	21 = hard tab
9 = home cursor	22 = clear line
10 = ring bell	23 = insert mode
11 = carriage return	24 = end insert mode
12 = inverse	25 = underline mode
13 = normal	26 = end underline mode
	27 = MouseText on
	28 = MouseText off

Input: prmtbl[0] = control code
prmtbl[1..2] = control arguments (GotoXY)
Output: prmtbl[0..1] = count of characters in buffer
prmtbl[2..3] = address of remote console control code buffer.

```
CTStatus        equ     4
```

CTStatus returns a flag describing a remote console's abilities to perform the specified control code. See CTControl for a list of control codes.
Input: prmtbl[0] = control code
Output: C=0 not serviceable
C=1 remote console can handle the control code

```
CTGetXY         equ     5
```

CTGetXY returns the cursor's current coordinates.
Input: None
Output: X=horizontal column
A=vertical row

```
CTWriteChar     equ     6
```

CTWriteChar writes a character to the console.
Input: prmtbl[0]=character
Output: None

```
CTWriteBuffer   equ     7
```

CTWriteBuffer writes a buffer of characters to the console.
Input: prmtbl[0..1]=count
prmtbl[2..3]=address of character buffer
Output: None

```
CTTestChar      equ     8
```

CTTestChar tests the keyboard to see if a character is waiting to be read with CTReadChar.
Input: None
Output: C=1 if a character is waiting to be read

```
CTReadChar      equ     9
```

CTReadChar reads the keyboard for a character. If one is available, it clears the keyboard. Note: This function does not wait indefinitely for a character—it returns immediately. It is different from CTTestChar in that it clears the keyboard of the character just read.
Input: None
Output: C=1 if a character is waiting to be read
A=character (with bit 7 set)

```
CTFlushInQ      equ     10
```

This function flushes the keyboard of any characters waiting to be read.
Input: None
Output: None

```
CTShowCursor    equ     11
CTHideCursor    equ     12
```

These functions show or hide the cursor character. Applications that allow the user to input information must manage the display of the cursor.
Input: None
Output: None

```
CTSetBellAttr equ     13
```

CTSetBellAttr sets the pitch and duration of the bell character.
Input: prmtbl[0]=pitch
prmtbl[1]=duration
Output: None

```
CTSetTermcap   equ     14
```

CTSetTermcap specifies the address of a terminal capability (termcap) structure. This structure defines the characteristics of a remote terminal for emulation.
Input: prmtbl[0..1] = address of termcap structure
Output: None

```
CTGotoXY       equ     15
```

This function places the cursor at the specified coordinates on the display.
Input: prmtbl[0]=horizontal column
prmtbl[1]=vertical row
Output: None

PrinterTool

Printer Tools handle output with a printer device. These tools provide their own port driver code, as well as support for specific kinds of printers they may drive.

```
************************************************
***
***    PrinterTool.equ
***

LT_ID          equ     $746C ;Printer Tool ("lt")
ID

LTOpen         equ     0
```

LTOpen initializes a session with a printer located in the slot specified.
Input: prmtbl[0]=slot of printer interface
Output: None

```
LTClose         equ     1
```

LTClose ends a session with a printer previously opened with LTOpen.
Input: None
Output: None

```
LTReset         equ     2
```

LTReset resets the printer previously opened with LTOpen.
Input: None
Output: None

```
LTControl       equ     3
```

This function performs various printer effects such as bold facing, and underlining, etc. Control codes have not yet been assigned. This function currently does nothing.
Input: prmtbl[0]=control code
prmtbl[1..n]=control code arguments
Output: None

```
LTWriteChar     equ     5
```

LTWriteChar writes a character to the printer.
Input: prmtbl[0]=character
Output: None

```
LTWriteBuffer equ     6
```

LTWriteBuffer writes a buffer of characters to the printer.
Input: prmtbl[0..1]=count
prmtbl[2..3]=address of character buffer
Output: None

SendTool

Send Tools perform file transfers using various communications protocols.

```
*************************************************
***
***     SendTool.equ
***

ST_ID           equ     $7473 ;Send Tool ("st") ID


STTransfer      equ     0
```

Sends a file using protocol. The address of the filename to transfer is stored in a pointer in the ProDOS BASIC global page at vpath1 ($BE6C). A string descriptor for a set of option characters is stored at lowtr ($9B). A null filename signifies the end of a batch transfer. If a disk error occurs, this function sets the carry flag and returns the error code in the A register. If carry is clear upon return, location a1 ($3C) contains a 16-bit transfer result code. A result of zero indicates a successful transfer—no errors.

Input: vpath1[0..1]=address of filename
lowtr[0]=option string length
lowtr[1..2]=address of option string

Output: If C=0 then a1[0..1]=transfer result
If C=1 then A=ProDOS BASIC error code

ReceiveTool

Receive Tools perform file transfers using various communications protocols.

```
*************************************************
***
***     ReceiveTool.equ
***

RT_ID           equ     $7472 ;Receive Tool ("rt")ID


RTTransfer      equ     0
```

Receives a file using protocol. The address of a filename in which to receive data is stored in a pointer in the ProDOS BASIC global page at vpath1 ($BE6C). A string descriptor for a set of option characters is stored at lowtr ($9B). If a disk error occurs, this function sets the carry flag and returns the error code in the A register. If carry is clear upon return, location a1 ($3C) contains a 16-bit transfer result code. A result of zero indicates a successful transfer—no errors. If the Receive Tool can ascertain the name of the file being sent, it returns it into a buffer pointed to by vpath2 ($BE6E). Return a null filename to signify the end of a batch transfer.

Input: vpath1[0..1]=address of filename
lowtr[0]=option string length
lowtr[1..2]=address of option string

Output: If C=0 then a1[0..1]=transfer result
If C=1 then A=ProDOS BASIC error code
vpath2[0..1]=address of returned filename

Sample Program

This chapter presents the source code for a custom Terminal Tool module called HexTerm. When HexTerm is used in place of the Terminal module, incoming data is displayed with hexadecimal values shown under each character. This makes debugging serial connections quite easy.

HexTerm

```
*******************************************************
***
***    hexterm.aii Hex Output Terminal Tool
***                Copyright (C) 1992 Morgan Davis
***                MPW IIGS Assembler
***

                MACHINE    M65C02
                longa      off
                longi      off
                case on

                INCLUDE    'OMM.equ'
                INCLUDE    'PortTool.equ'

a1              equ  $3c
prmtbl          equ  $e0
chrgot          equ  $b7

ch80            equ  $057b

kbd             equ  $c000
strb            equ  $c010
cmdkey          equ  $c061

chkcom          equ  $debe
getbyte         equ  $e6f8
bs              equ  $fc10
up              equ  $fc1a
lf              equ  $fc66
prbyte          equ  $fdda
cout            equ  $fded
```

```
term        PROC
hVERS       DC.W $0000          ;OMM header
hID         DC.W 'tm'
hSIZE       DC.W END-START
hORG        DC.W START
hAMPC       DC.W amperc
hKIND       DC.W $0000
hRSRV1      DC.W $0000
hRSRV2      DC.W $0000

START       cmp  #MSG_AMPR      ;ampersand call?
            beq  doampr         ;yes
            cmp  #MSG_DIED      ;module death?
            beq  dodeath
            cmp  #MSG_BORN      ;module birth?
            beq  dobirth
            cmp  #MSG_INFO      ;get info string?
            bne  ctrts

doinfo      lda  a_info
            sta  a1
            lda  a_info+1
            sta  a1+1
ctrts       rts

callpt      ldx  ptindex        ;function in Y
            beq  ctrts          ;oops no tool!
dommvec     jmp  OMMVEC         ;call the Port Tool

dobirth
dodeath
            lda  #<PT_ID        ;get port tool index
            sta  a1
            lda  #>PT_ID
            sta  a1+1
            jsr  ommid
            stx  ptindex        ;save it
            rts

ommid       ldx  #OMM_ID
            ldy  #OMM_GETID
            jmp  OMMVEC
```

```
*------------------------------*
* Ampersand Command Dispatcher *
*------------------------------*

doampr        jsr  chrgot        ;any arguments?
              beq  termread      ;no

              jsr  getbyte       ;get flag in X

*------------------------------*
*      Terminal Mode Loop      *
*------------------------------*

termread      jsr  updateTerm    ;update display
              ldy  #SerGetDCD    ;check DCD
              jsr  callpt
              bcc  tmnocar       ;none, so quit

keyread       lda  kbd           ;check keyboard
              bpl  termread

              bit  cmdkey        ;check Command key
              sta  strb          ;clear keyboard
              bmi  tmq           ;got a command

              and  #$7F          ;send key to port
              sta  prmtbl
              ldy  #SerWriteChar
              jsr  callpt
              bra  termread      ;loop back for more
```

```
*------------------------------*
*      Exit Terminal Mode      *
*------------------------------*

tmnocar       lda  #0
tmq           sta  result       ;save exit code
              jsr  chrgot       ;return it?
              beq  elret        ;no

              jsr  chkcom       ;skip comma
              lda  result
              sta  a1
              stz  a1+1
              ldy  #OMM_PUTWORD
              ldx  #OMM_ID
              jmp  OMMVEC       ;return result code

*------------------------------*
*        Update Display        *
*------------------------------*

updateTerm    ldy  #SerReadChar ;serial input?
              jsr  callpt
              bcs  termout      ;yes

elret         rts

termout       pha
              ora  #$80         ;make it printable
              cmp  #$A0         ;check for control
              bcs  tocout

              and  #$7F         ;invert controls
              ora  #$40
tocout        jsr  cout         ;print character
              jsr  bs           ;back up
              jsr  lf           ;go down
              pla               ;get real byte
              jsr  prbyte       ;print it in hex
              lda  ch80         ;check for wrap
              beq  elret

              jmp  up           ;no wrap, so go up
```

```
*------------------------------*
*         Data Section         *
*------------------------------*

              dc.b   $00          ;start of immed
table
a_info        dc.w   info
              dc.w   $0000        ;start of data

amperc        dc.b   'TERM',0     ;&TERM invokes this
              dc.b   -1

              msb    on
info          dc.b   '18-May-92 HexTerm 1.0'
              msb    off

ptindex       ds.b   1            ;index to Port Tool
result        ds.b   1            ;exit code

END
              ENDP

              END
```

ASCII Chart

Low			High		Low			High		Low			High		Low			High	
0	$00	^@	128	$80	32	$20	SPC	160	$A0	64	$40	@	192	$C0	96	$60	‘	224	$E0
1	$01	^A	129	$81	33	$21	!	161	$A1	65	$41	A	193	$C1	97	$61	a	225	$E1
2	$02	^B	130	$82	34	$22	"	162	$A2	66	$42	B	194	$C2	98	$62	b	226	$E2
3	$03	^C	131	$83	35	$23	#	163	$A3	67	$43	C	195	$C3	99	$63	c	227	$E3
4	$04	^D	132	$84	36	$24	$	164	$A4	68	$44	D	196	$C4	100	$64	d	228	$E4
5	$05	^E	133	$85	37	$25	%	165	$A5	69	$45	E	197	$C5	101	$65	e	229	$E5
6	$06	^F	134	$86	38	$26	&	166	$A6	70	$46	F	198	$C6	102	$66	f	230	$E6
7	$07	^G	135	$87	39	$27	'	167	$A7	71	$47	G	199	$C7	103	$67	g	231	$E7
8	$08	^H	136	$88	40	$28	(	168	$A8	72	$48	H	200	$C8	104	$68	h	232	$E8
9	$09	^I	137	$89	41	$29	)	169	$A9	73	$49	I	201	$C9	105	$69	i	233	$E9
10	$0A	^J	138	$8A	42	$2A	*	170	$AA	74	$4A	J	202	$CA	106	$6A	j	234	$EA
11	$0B	^K	139	$8B	43	$2B	+	171	$AB	75	$4B	K	203	$CB	107	$6B	k	235	$EB
12	$0C	^L	140	$8C	44	$2C	,	172	$AC	76	$4C	L	204	$CC	108	$6C	l	236	$EC
13	$0D	^M	141	$8D	45	$2D	-	173	$AD	77	$4D	M	205	$CD	109	$6D	m	237	$ED
14	$0E	^N	142	$8E	46	$2E	.	174	$AE	78	$4E	N	206	$CE	110	$6E	n	238	$EE
15	$0F	^O	143	$8F	47	$2F	/	175	$AF	79	$4F	O	207	$CF	111	$6F	o	239	$EF
16	$10	^P	144	$90	48	$30	0	176	$B0	80	$50	P	208	$D0	112	$70	p	240	$F0
17	$11	^Q	145	$91	49	$31	1	177	$B1	81	$51	Q	209	$D1	113	$71	q	241	$F1
18	$12	^R	146	$92	50	$32	2	178	$B2	82	$52	R	210	$D2	114	$72	r	242	$F2
19	$13	^S	147	$93	51	$33	3	179	$B3	83	$53	S	211	$D3	115	$73	s	243	$F3
20	$14	^T	148	$94	52	$34	4	180	$B4	84	$54	T	212	$D4	116	$74	t	244	$F4
21	$15	^U	149	$95	53	$35	5	181	$B5	85	$55	U	213	$D5	117	$75	u	245	$F5
22	$16	^V	150	$96	54	$36	6	182	$B6	86	$56	V	214	$D6	118	$76	v	246	$F6
23	$17	^W	151	$97	55	$37	7	183	$B7	87	$57	W	215	$D7	119	$77	w	247	$F7
24	$18	^X	152	$98	56	$38	8	184	$B8	88	$58	X	216	$D8	120	$78	x	248	$F8
25	$19	^Y	153	$99	57	$39	9	185	$B9	89	$59	Y	217	$D9	121	$79	y	249	$F9
26	$1A	^Z	154	$9A	58	$3A	:	186	$BA	90	$5A	Z	218	$DA	122	$7A	z	250	$FA
27	$1B	^[	155	$9B	59	$3B	;	187	$BB	91	$5B	[	219	$DB	123	$7B	{	251	$FB
28	$1C	^\	156	$9C	60	$3C	<	188	$BC	92	$5C	\	220	$DC	124	$7C	|	252	$FC
29	$1D	^]	157	$9D	61	$3D	=	189	$BD	93	$5D	]	221	$DD	125	$7D	}	253	$FD
30	$1E	^^	158	$9E	62	$3E	>	190	$BE	94	$5E	^	222	$DE	126	$7E	~	254	$FE
31	$1F	^_	159	$9F	63	$3F	?	191	$BF	95	$5F	_	223	$DF	127	$7F	DEL	255	$FF

Low High Low High Low High Low High

ProDOS File Types

Type	Hex	Dec	Description
UNK	$00	0	Unknown
BAD	$01	1	Bad Blocks
PCD	$02	2	Apple /// Pascal Code
PTX	$03	3	Apple /// Pascal Text
TXT	$04	4	ASCII Text
PDA	$05	5	Apple /// Pascal Data
BIN	$06	6	General Binary
FNT	$07	7	Apple /// Font
FOT	$08	8	Graphics
BA3	$09	9	Apple /// BASIC Program
DA3	$0A	10	Apple /// BASIC Data
WPF	$0B	11	Word Processor
SOS	$0C	12	Apple /// SOS System
DIR	$0F	15	Folder
RPD	$10	16	Apple /// RPS Data
RPI	$11	17	Apple /// RPS Index
AFD	$12	18	Apple /// AppleFile Discard
AFM	$13	19	Apple /// AppleFile Model
AFR	$14	20	Apple /// AppleFile Report Format
SCL	$15	21	Apple /// Screen Library
PFS	$16	22	PFS Document
ADB	$19	25	AppleWorks Data Base
AWP	$1A	26	AppleWorks Word Processor
ASP	$1B	27	AppleWorks Spread Sheet
TDM	$20	32	Desktop Manager Document
8SC	$29	42	Apple II Source Code
8OB	$2A	43	Apple II Object Code
8IC	$2B	44	Apple II Interpreted Code
8LD	$2C	45	Apple II Language Data
P8C	$2D	46	ProDOS 8 Code Module
FTD	$42	66	File Type Names
GWP	$50	80	Apple IIGS Word Processor
GSS	$51	81	Apple IIGS Spread Sheet
GDB	$52	82	Apple IIGS Data Base
DRW	$53	83	Drawing
GDP	$54	84	Desktop Publishing
HMD	$55	85	Hypermedia
EDU	$56	86	Educational Data
STN	$57	87	Stationery
HLP	$58	88	Help
COM	$59	89	Communications
CFG	$5A	90	Configuration
ANM	$5B	91	Animation
MUM	$5C	92	Multimedia
ENT	$5D	93	Entertainment
DVU	$5E	94	Development Utility

Continued . . .

ProDOS File Types (Continued)

Type	Hex	Dec	Description
BIO	$6B	107	PC Transporter BIOS
TDR	$6D	109	PC Transporter Driver
PRE	$6E	110	PC Transporter Pre-Boot
HDV	$6F	111	PC Transporter Volume
WP	$A0	160	WordPerfect Document
GSB	$AB	171	Apple IIGS BASIC Program
TDF	$AC	172	Apple IIGS BASIC TDF
BDF	$AD	173	Apple IIGS BASIC Data
SRC	$B0	176	Apple IIGS Source
OBJ	$B1	177	Apple IIGS Object
LIB	$B2	178	Apple IIGS Library
S16	$B3	179	GS/OS Application
RTL	$B4	180	GS/OS Run-time Library
EXE	$B5	181	GS/OS Shell Application
PIF	$B6	182	Permanent Initialization
TIF	$B7	183	Temporary Initialization
NDA	$B8	184	New Desk Accessory
CDA	$B9	185	Classic Desk Accessory
TOL	$BA	186	Tool
DRV	$BB	187	Device Driver
LDF	$BC	188	Load File
FST	$BD	189	GS/OS File System Translater
DOC	$BF	191	GS/OS Document
PNT	$C0	192	Packed Super Hi-Res Picture
PIC	$C1	193	Super Hi-Res Picture
ANI	$C2	194	Animation
PAL	$C3	195	Palette
OOG	$C5	197	Object Oriented Graphics
SCR	$C6	198	Script
CDV	$C7	199	Control Panel
FON	$C8	200	Font
FND	$C9	201	Finder Data
ICN	$CA	202	Icons
MUS	$D5	213	Music Sequence
INS	$D6	214	Instrument
MDI	$D7	215	MIDI
SND	$D8	216	Sampled Sound
DBM	$DB	219	Relational Data Base File
LBR	$E0	224	Archival Library
ATK	$E2	226	AppleTalk Data
R16	$EE	238	EDASM 816 Relocatable File
PAS	$EF	239	Pascal Area
CMD	$F0	240	BASIC Command
LNK	$F8	248	EDASM Linker
OS	$F9	249	GS/OS System File
INT	$FA	250	Integer BASIC Program
IVR	$FB	251	Integer BASIC Variables
BAS	$FC	252	Applesoft BASIC Program
VAR	$FD	253	Applesoft BASIC Variables
REL	$FE	254	Relocatable Code
SYS	$FF	255	ProDOS 8 System Application

Error Codes

0 **NEXT Without FOR**: a NEXT was encountered which had no matching FOR.
2 **Range Error**: an invalid argument value was specified.
3 **No Device Connected**: the given slot has no disk drive installed.
4 **Write Protected Disk**: unable save data unless write-enabled.
5 **End of Data**: an attempt was made to read data past the end of a file.
6 **Path Not Found**: the path to a filename was not found.
7 **File Not Found**: the specified file was not found.
8 **I/O Error**: the drive went offline or the disk has a media defect.
9 **Disk Full**: no room exists on the disk storing more data.
10 **File Locked**: the file is protected against modification or removal.
11 **Invalid Option**: an option not allowed for a certain command was used.
12 **No Buffers Available**: not enough memory for further disk functions.
13 **File Type Mismatch**: an invalid attempt was made to access a special file.
14 **Program Too Large**: you've written a FAT and SLOPPY program.
15 **Not Direct Command**: command was issued from immediate mode.
16 **Syntax Error**: a filename is illegal or a program statement misspelled.
17 **Directory Full**: the root volume contains too many filenames.
18 **File Not Open**: an attempt was made to read or write from an closed file.
19 **Duplicate File Name**: a RENAME or CREATE used on an existing filename.
20 **File Busy**: an attempt to re-OPEN or modify an OPEN file's name was made.
21 **File Still Open**: upon entering immediate mode, a file was found OPEN.
22 **RETURN Without GOSUB**: a RETURN with no matching GOSUB.
42 **Out of Data**: an attempt was made to READ past the last DATA item.
53 **Illegal Quantity**: an out-of-range value was used with a certain command.
69 **Overflow**: you used an awfully BIG or amazingly SMALL number.
77 **Out of Memory**: program code and variables have used up all free memory.
90 **Undef'd Statement**: a line number which does not exist was referenced.
107 **Bad Subscript**: an array subscript is larger than the given DIMension.
120 **Redim'd Array**: an attempt was made to reDIMension an existing array.
133 **Division by Zero**: division by zero is undefined (remember your algebra?)
163 **Type Mismatch**: a numeric or string value was used incorrectly.
176 **String Too Long**: the given string was larger than was allowed.
191 **Formula Too Complex**: go easy on the machine, Einstein.
224 **Undef'd Function**: reference to an undefined FuNction was made.
254 **Reenter**: user input was not of the type or format required.
255 **Control-C Interrupt**: control-C was pressed.