NISTIR 6611

Technical Reference Manual for NIST Automated Computer Time Service (ACTS)

Judah Levine
Michael A. Lombardi
Lisa M. Nelson
Victor S. Zhang

NIST
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

NISTIR 6611

Technical Reference Manual for NIST Automated Computer Time Service (ACTS)

Judah Levine
Michael A. Lombardi
Lisa M. Nelson
Victor S. Zhang
Time and Frequency Division
Physics Laboratory
325 Broadway
Boulder, CO 80305

July 2001

U.S. Department of Commerce
Donald L. Evans, Secretary

National Institute of Standards and Technology
Karen H. Brown, Acting Director

Table of Contents

Introduction to ACTS Server and Monitor

The NIST Automated Computer Time Service (ACTS) allows users with analog modems and client software to synchronize their clocks to NIST time. This manual documents the system used by NIST to provide the service. When properly installed and operated, the system described in this manual can be used to provide a dial-up service identical to ACTS at any location.

The ACTS system consists of two main components, the server and the monitor. The server is a PC-based device that can handle up to four incoming calls at one time. All callers who connect to the server are sent a synchronized ASCII time code. The monitor is also a PC-based device with an independent reference clock. The monitor continually queries the server (through the serial and telephone lines) and reports time code errors or server hardware failures.

This manual is a comprehensive overview of the ACTS system. Chapter 1 is a general overview of the information ACTS provides to its users. Chapters 2 and 3 describe how to install and operate the server and monitor. Chapters 4 and 5 provide a technical reference for the server and monitor.

Chapter 1

Overview of ACTS

The poor performance of computer clocks is well documented, and causes problems in many areas of business, government, and industry. Many computer applications require that time be kept to the nearest second or better, and existing computer hardware is simply not up to the task. For example, the computers in a financial institution must keep very accurate records of when transactions were completed, for legal or other reasons. Computer systems that make physical measurements and acquire scientific data need to know precisely when the measurements were made. Software used on a manufacturing floor may need to turn a piece of equipment on or off at a specified time. Also, any system involved with synchronous communications must keep accurate time. For example, radio and TV stations may need computers that can switch feeds or link up with remotes at the right time.

In response to these problems, NIST began the Automated Computer Time Service (ACTS) in 1988. This service allows any user with the appropriate client software and modem to synchronize their computer clock using an ordinary telephone line. This manual documents the dial-up time server and monitoring system used by ACTS. This chapter provides an overview of what ACTS is, and how ACTS works. Let's begin our discussion by describing how a typical computer keeps time.

How a Personal Computer Keeps Time

Since the introduction of the IBM-AT personal computer in 1984, all PC-compatible computers have kept time the same way. Each PC built using the AT architecture contains two clocks, regardless of whether it uses the 286, 386, 486, or Pentium microprocessor (or a derivative). These clocks go by several different names, but for simplicity, we'll call them the software and hardware clocks. The software clock keeps time while the computer is turned on, but stops when the computer is turned off. The hardware clock has its own power supply (battery) and runs even while the computer is turned off.

The software clock is generated by an Intel 8254 timer-counter (or a functionally equivalent device). This timer-counter generates an interrupt every 54.936 ms, or about 18.2 times per second. The computer's BIOS (Basic Input Output System) contains a software routine that counts the interrupt requests and generates a time-of-day clock that can be read or set by

other software programs. For example, DOS uses time-of-day information from the software clock to date and time stamp files.

The software clock is a poor timekeeper. Its timing uncertainty is limited by the stability of the interrupt requests. Any change in the interrupt request rate causes the clock to gain or lose time. If you leave your computer turned on for long periods, the software clock might be off by large amounts, perhaps a minute or more for every day that the computer was left turned on. It's also possible for an ill-behaved software program to use the timer-counter for another purpose and change its interrupt rate. This could cause the clock to rapidly gain or lose time.

The software clock also has limited resolution. It can display only values that are even multiples of the time interval between interrupts (55 ms). For example, 00:00:01.00 could never be displayed by the software clock. The closest possible values it can display are 00:00:00.98 and 00:00:01.04.

The single biggest drawback of the software clock, however, is that when the computer is turned off, the clock stops running and loses all of its time-of-day information. For this reason, a hardware clock is also necessary. The hardware clock is based on the Motorola 146818 Real Time Clock Chip, or an equivalent device. When the computer is turned off, the hardware clock runs off batteries. When the computer is turned back on, the software clock starts running again and sets itself (within 1 s) to the hardware clock. Although the hardware and software clocks are synchronized at power-up, they run at different rates and will gain or lose time relative to each other while the computer is running.

The hardware clock is updated once per second and cannot display fractions of a second. Its timing uncertainty is determined by the quality of the crystal oscillator it uses as its time base. These crystals cost less than $1 in single quantities and offer only marginal timekeeping performance. They are sensitive to temperature and other factors and their frequency uncertainty is unlikely to be better than 1×10^{-5} (about 1 s per day). In actual operation, most hardware clocks gain or lose about 5 to 15 s per day, with 10 s per day being typical. Although the hardware clock usually outperforms the software clock, its performance pales in comparison to even a low-cost wristwatch.

As you can see, neither the software or hardware clock is suitable for accurate timekeeping. This is why ACTS and other dial-up services are so valuable. They allow users to synchronize computer clocks with a simple telephone call.

Using ACTS

Using ACTS requires only a computer, a modem, and some simple software. When a computer connects to ACTS by telephone, it receives an ASCII time code. The information in this time code is then used to set the computer clock to the correct time. ACTS is usable at modem speeds up to 9600 baud with 8 data bits, 1 stop bit, and no parity. To receive the full time code, you must connect at a speed of at least 1200 baud. The full time code is transmitted every second and contains more information than the 300 baud time code, which is transmitted once every 2 s. The table describes the full ACTS time code:

JJJJJ YR-MO-DA HH:MM:SS TT L UT1 msADV UTC(NIST) <OTM>
JJJJJ is the Modified Julian Date (MJD). The MJD is the last five digits of the Julian Date, which is simply a count of the number of days since January 1, 4713 B.C. To get the Julian Date, add 2.4 million to the MJD.
YR-MO-DA is the date. It shows the last two digits of the year, the month, and the current day of month.
HH:MM:SS is the time in hours, minutes, and seconds. The time is always sent as Coordinated Universal Time (UTC). An offset needs to be applied to UTC to obtain local time. For example, Mountain Time in the U. S. is 7 hours behind UTC during Standard Time, and 6 hours behind UTC during Daylight Saving Time.
TT is a two digit code (00 to 99) that indicates whether the United States is on Standard Time (ST) or Daylight Saving Time (DST). It also indicates when ST or DST is approaching. This code is set to 00 when ST is in effect, or to 50 when DST is in effect. On the day of the transition from DST to ST, the code is set to 01. On the day of the transition from ST to DST, the code is set to 51. The client software is responsible for implementing the change at 2 a.m. on the day of the transition. During the month of the transition, the code is decremented every day until the change occurs. For example, October is the month of the transition (in the United States) from DST to ST. On October 1, the number changes from 50 to the actual number of days until the time change. It will decrement by 1 every day, and reach 01 on the day of the transition. It will be set to 00 the day after the transition, and will remain there until the following April.
L is a one-digit code that indicates whether a leap second will be added or subtracted at midnight on the last day of the current month. If the code is 0, no leap second will occur this month. If the code is 1, a positive leap second will be added at the end of the month. This means that the last minute of the month will contain 61 seconds instead of 60. If the code is 2, a second will be deleted on the last day of the month. Leap seconds occur at a rate of about one per year. They are used to correct for irregularity in the earth's rotation.
UT1 is a correction factor for converting UTC to an older form of universal time. It is always a number ranging from -0.9 to +0.9 s. This number is added to UTC to obtain UT1.
msADV is a five-digit code that displays the number of milliseconds that NIST advances the time code. It is originally set to 45.0 ms. If you return the on-time marker (OTM) three consecutive times, it will change to reflect the actual one-way line delay.
The label UTC(NIST) is contained in every time code. It indicates that you are receiving Coordinated Universal Time (UTC) from the National Institute of Standards and Technology (NIST).
The on-time marker (OTM) is a single character sent at the end of each time code. The OTM is originally an asterisk (*) and changes to a pound sign (#) if ACTS has successfully calibrated the path delay.

The last character in the time code is an asterisk (*). The asterisk is called the On-Time Marker (OTM). The time values sent by the time code refer to the arrival time of the OTM. In other words, if the time code says it is 12:45:45, this means it is 12:45:45 when the OTM arrives. Of course, there is some delay between the time the OTM leaves the ACTS server and the time it arrives at your computer. Some of this delay is the actual data transmission time. However, most of the delay occurs when the modem processes the incoming data and sends it to the computer.

Since there is some path delay as the OTM travels from ACTS to your computer, the OTM is sent out 45 ms early. The 45 ms advance was chosen based on experiments conducted at NIST using 1200 baud modems. This 45 ms includes the 8 ms that it takes to send the OTM at 1200 baud, 7 ms transmission time to allow for travel from NIST to an average user in the United States, and 30 ms to allow for the modem differential delay. At 9600 baud, OTM transmission time drops to 1 ms, but modem differential delay becomes less stable due to software data compression routines and could be much larger than 30 ms.

Advancing the OTM by 45 ms always removes some of the delay. However, to get the least amount of timing uncertainty, we need to advance the OTM by the amount of the actual path delay. ACTS can do this by using a *loop-back* technique to calibrate the path. The loop-back technique works if the user's computer software returns the OTM to NIST after it is received. Each time the OTM is returned, ACTS measures the amount of time it took for the OTM to go from the server to the user and back to the server. This quantity is the round-trip path delay so it is divided by 2 to get the one-way path delay. After three consecutive measurements have been made, the server advances the time by the amount of the one-way path delay. For example, if the one-way path delay is 50.4 ms, the server sends the OTM out 50.4 ms (instead of 45 ms) early. At this point, the path delay is calibrated, and OTM changes from an asterisk to a pound sign (#). If you calibrate the path delay, ACTS can set a computer clock with an uncertainty of less than 5 ms.

To illustrate how simple ACTS software can be, a rudimentary ACTS program is listed on the next few pages. This software is written in QBASIC, the language included with MS-DOS (versions 5.0 and higher). It should work on any PC-compatible. The software dials the ACTS server at NIST in Boulder, Colorado, sends the OTM back to NIST and calibrates the path, and uses the QBASIC functions DATE$ and TIME$ to set the date and time on the computer clock.

The software has limitations. For example, it can set a clock only to UTC. A routine must be added if you need to convert UTC to local time. Also, it works only with COM1 or COM2 (and not COM3 or COM4) since only these serial ports are supported by QBASIC.

```
DECLARE SUB StripCRLF (x$)
' ************************************************************************
' Software to access the NIST Automated Computer Time Service
' (Works with QBASIC compiler included with MS-DOS 5.0 and later versions)
' ************************************************************************
'       Define Modem Initialization String
' ************************************************************************

        attention$ = "ATZX3"

' ************************************************************************
'       Define number of time codes to receive
' ************************************************************************

        receivelines% = 10

' ************************************************************************
'       COM Port Setting (set to either COM1: or COM2:)
' ************************************************************************

        comport$ = "COM2:"

' ************************************************************************
'       Baud Rate Setting (set to 9600, 2400, 1200, or 300 baud)
' ************************************************************************

        baudrate$ = "1200"

' ************************************************************************
'       Dialing Method (set to T for tone dialing, P for pulse dialing)
' ************************************************************************

        pulse$ = "T"

' ************************************************************************
'       Phone Number (eliminate area code, add access codes if necessary)
' ************************************************************************
        phone$ = "1-303-494-4774"
```

```
' ************************************************************************
'      housekeeping functions
' ************************************************************************

        ON ERROR GOTO Handler                       ' trap errors
        CLS                                     ' clear screen
        IF VAL(baudrate$) > 300 THEN codefix% = 6 ELSE codefix% = 0

' ************************************************************************
'      print opening text on screen
' ************************************************************************

        PRINT "NIST Automated Computer Time Service (ACTS)"
        PRINT
        PRINT "Dialing ACTS, please wait ..."
        PRINT

' ************************************************************************
'      set up modem for 8 data bits, 1 stop bit, and no parity
' ************************************************************************

        communicate$ = comport$ + baudrate$ + ",N,8,1"

' ************************************************************************
'      call the Automated Computer Time Service (ACTS)
' ************************************************************************

        OPEN communicate$ FOR RANDOM AS #1 LEN = 256

                PRINT #1, attention$                ' initialize modem
                g = TIMER
                DO
                LOOP UNTIL ABS(TIMER - g) > 3           ' pause for 3 seconds
                PRINT #1, "ATD" + pulse$ + phone$       ' dial number
                printline% = 1                          ' count first line

        DO

                timecode$ = ""                          ' clear timecode$
```

```
            DO                                          ' read in characters
                  s$ = INPUT$(1, #1)
                  timecode$ = timecode$ + s$
            LOOP UNTIL s$ = "#" OR s$ = "*" OR s$ = CHR$(13)
            CALL StripCRLF(timecode$)                   ' clean up time code

            SELECT CASE timecode$                       ' check for errors
                  CASE "ERROR", "NO DIALTONE", "BUSY", "NO CARRIER",
"NO ANSWER"
                        CLOSE #1
                        GOTO Handler
            END SELECT

            otm$ = s$                                   ' check for on-time marker

            IF otm$ = "*" OR otm$ = "#" THEN

                  PRINT #1, otm$                        ' send OTM back to NIST
                  PRINT timecode$                       ' print time code on screen
                  printline% = printline% + 1           ' increment print line

            END IF

      LOOP UNTIL printline% > receivelines%             ' get next time code

      T$ = MID$(timecode$, 10 + codefix%, 8)            ' get time from time code
      TIME$ = T$                                        ' set time on computer
      d$ = MID$(timecode$, 4 + codefix%, 5) + "-"       ' get date from time code
      d$ = d$ + “20” + MID$(timecode$, 1 + codefix%, 2)
      DATE$ = d$                                        ' set date on computer
      PRINT #1, "ATH"                                   ' hang up telephone
      CLOSE #1

      PRINT
      PRINT "The computer's clock has been set."        ' print message

      END
```

```
' ************************************************************************
Handler:                                    ' print error message

        PRINT
        PRINT "Communications Error  -  Check Phone Line, Modem, and"
        PRINT "                     COM port setting."
        PRINT

END
' ************************************************************************

SUB StripCRLF (x$) STATIC

        ' strips all carriage returns and line feeds from a string

        y$ = ""
        FOR i% = 1 TO LEN(x$)
                z$ = MID$(x$, i%, 1)
                IF ASC(z$) <> 10 AND ASC(z$) <> 13 THEN
                        y$ = y$ + z$
                END IF
        NEXT i%
        x$ = y$

END SUB
```

There are a number of more sophisticated software packages available both commercially and through shareware that call ACTS. Any ACTS compatible software package will be able to call the servers described in this manual.

Chapter 2

The ACTS Dial-Up Time Server

This chapter provides a brief overview of the ACTS dial-up time server. It provides a parts list for the server, and explains how to install, start, and operate the server.

Parts List

The ACTS server consists of the following parts:

- Rack Mount Computer

Any 486 or Pentium computer (33 MHz or faster) running the DOS operating system should work. Both Microsoft MS-DOS 6.22 and Datalight ROMDOS 6.22 have been successfully tested. The computer requires any video card capable of displaying text, 4 MB of RAM, a 3.5 inch floppy drive, and a hard disk or electronic flash disk. Depending on the configuration, the computer might need as many as 5 ISA bus slots. At this writing (2001) it might be necessary to use a passive backplace design with a CPU card, since motherboards with more than two ISA bus slots are difficult to find.

The computer should be used with a rack mount keyboard and monitor (VGA). Some industrial computers have a VGA monitor (CRT or LCD display) and/or a keyboard built-in. These should work fine.

- GT401 Event/Timing Controller, manufactured by Guide Technology Inc.

This is a full length ISA bus card that accepts the external timing reference for the ACTS server. The server software has been designed to use this card. Guide Technology can be reached at (408) 733-6555 or on the web at www.guidetech.com.

- Multiple serial ports

As many as six serial ports can be used with the ACTS server. Any serial port board that is compatible with DOS and allows you to select a unique address and IRQ for each port should work. Boards that “cascade” multiple serial ports using a single IRQ are not compatible with the existing ACTS server software.

- Ethernet board (if network verify option is installed)

Any ethernet board that includes drivers for DOS should work.

- As many as six external modems

The number of modems should match the number of serial ports. ACTS has been successfully tested with US Robotics Sportster modems running at 14.4, 28.8, and 33.6 kilobits per second. Other modems will work with some minor software modifications.

- Serial cables, telephone cables, connectors/terminators and various spare parts

Installing the Server

The server must be installed in a location accessible to four telephone lines and an on-time 1 pulse per second (pps) signal. Install the server as follows:

- Mount the server in an equipment rack. Situate the equipment so that the video display is located at a convenient viewing height, and so that the keyboard is at a convenient typing height. The top of the computer chassis makes a convenient resting place for the modems, so leave a gap between the bottom of the monitor shelf and the top of the computer chassis.

- Connect the video display (VGA monitor) to the 15-pin video jack on the back of the computer chassis. Plug the monitor's power cord into an AC outlet. Turn the monitor ON.

- Plug the keyboard into the round keyboard jack on the back of the computer chassis.

- Connect a serial cable from serial port 1 of the server to the monitor eavesdrop box. Then, connect a cable from the monitor eavesdrop box to the modem for port 1. Connect a serial cable from the remaining serial ports directly to the remaining external modems. The modems can be placed on top of the computer chassis. Connect each modem to the AC power, and to a telephone line using the supplied phone cables. Turn each modem ON.

- Connect your 1 pps time reference to the BNC connector on the back of the computer chassis (bottom connector) using a 50 W termination on a T-connector.

Skip this step if you will not use network verify:

- Connect your network connection to the individual ethernet card BNC connector using a T-connector with 50 W termination on the last connected server. A green light will blink when the connection has been established to the network card.

- Connect a power cord to the computer chassis.

- Turn the server on. The software has been pre-installed and will load as described in the next section.

Starting the Server

When the server is turned on, it will go through the following sequence.

- If the ACTS server is not connected to the Internet, the operator is prompted to manually enter the correct date and time (UTC). If the computer is connected to the Internet and the network verify option has been installed, DOS drivers for the Ethernet card are loaded, and the server will then synchronize its internal clock to the NIST Internet Time Service.

- The ACTS Directory is selected using this command:

CD\ACTS

- The Guide Technology board is initialized and synchronized using this command. The A100 parameter represents the I/O address of the Guide board (100h).

CLOCK401 /DOSTO401 /a100

- All modems are initialized with this command (for a six modem system):

6PORT /auto

The 6PORT software will initialize the modem connected to each serial port (software called FOURPORT is included with four modem systems, the modem initialization is described in detail in Chapter 5). When all modems have been initialized, the ACTS server software is loaded, and a screen appears like the one shown on the next page.

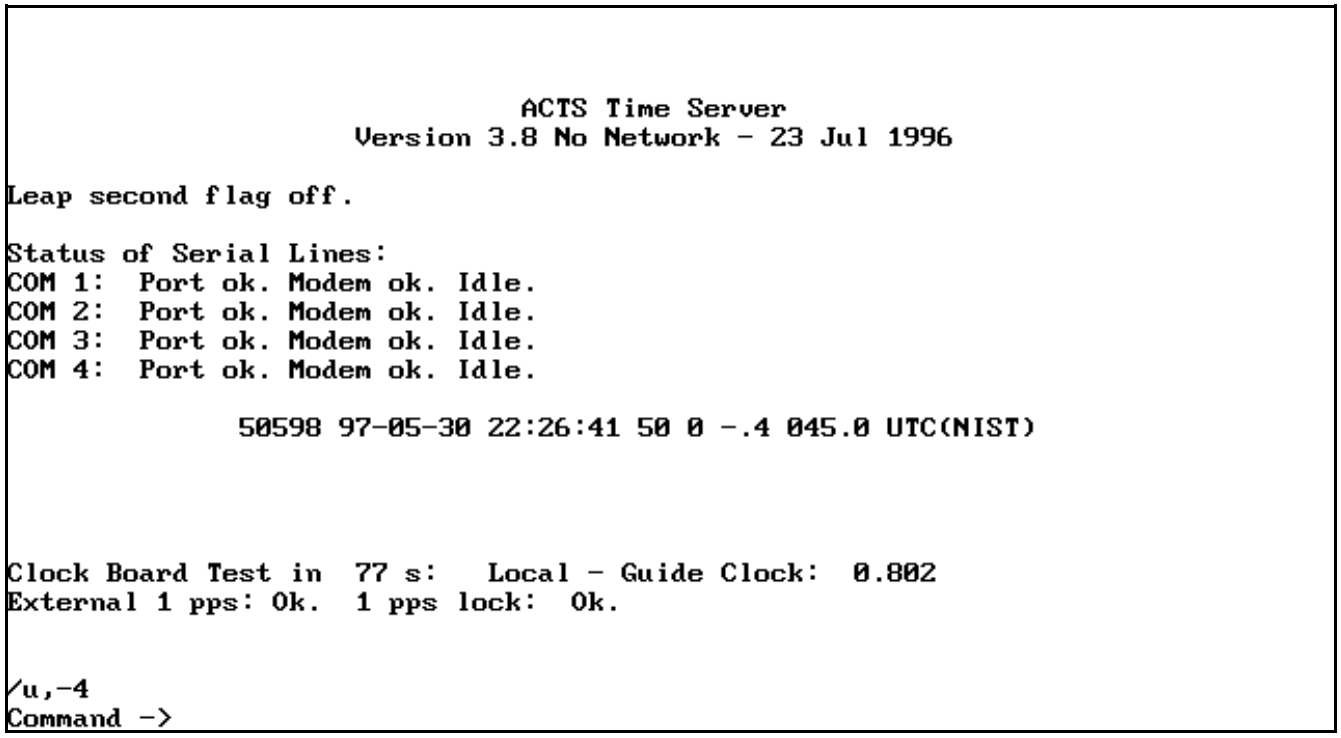

Figure 1. The ACTS Time Server Display

- There are several variations of the ACTS server software used for different applications. The software that is loaded will determine the capabilities of the ACTS server. The primary differences are the number of modems the software supports (four or six), whether or not the software includes the network verify option, and whether the software communicates at a maximum speed of 1200 or 9600 baud. The software pictured in Figure 1 supports four modems (note labels for COM1 through COM4), does not support the network, and communicates at speeds up to 9600 baud.

- Once the ACTS software loads, it will check all modems and serial ports to see if there are any problems. If not, the "Port Ok" and "Modem Ok" messages will appear on screen.

- Once all modems and ports have been verified as OK, add the UT1 correction to the time code by typing:

  /u, ±x

  where ± is the sign of the correction and x is the value (tenths of seconds). The default UT1 correction is +0.1 s.

- Check the status of the 1 pps connected to the ACTS server. There are three on-screen indicators you should check:

  ⇒ The "External 1 pps" indicator should be set to Ok.

  ⇒ The "1 pps lock" indicator should be set to Ok.

  ⇒ The "Local - Guide Clock" indicator shows the difference between the computer clock and the Guide Technology Board. Since ACTS was designed (Chapter 5) to have the computer clock set 13 to 15 timer ticks ahead of the Guide board, this difference should be a positive number between 0.715 and 0.825 s.

- If your server has the network verify option, check for network verify "Ok" on both nodes. If your software supports the network verify, but you are not going to use it, turn it off by typing **/v, -1** and **/v, -2** from the command prompt.

- Check the leap second flag indicator. It should normally be off, unless a leap second is scheduled at the end of the current month.

Using the Server

Once the server is operational, there are only a few things you need to do:

- Check the time code string for correct date and time.

- Check and set the UT1 correction when necessary by typing **/u, +x** from the command prompt (where + is the sign of the correction, and x is the value in tenths of seconds.

- Set the leap second indicator on the first day of the month when a leap second is scheduled to occur. Type **/l, 1** from the command prompt to add a second, or **/l, 2** to delete a second on the last day of the month. Use **/l, 0** to disable the leap second flag.

- To stop the server software, type **/h** at the command prompt.

- To update the display (to clear an error message), type **/d** to refresh the screen.

Always leave the keyboard locked so that the time code cannot be changed by unauthorized personnel.

Chapter 3

The ACTS Monitor

This chapter provides a brief overview of the ACTS monitor. It provides a parts list for the monitor, and explains how to install, start, and operate the monitor.

Parts List

The ACTS monitor consists of the following parts:

- Rack Mount Computer

The monitor computer can be the same type used for the server. Any 486 or Pentium computer (33 MHz or faster) running the DOS operating system should work. Both Microsoft MS-DOS 6.22 and Datalight ROMDOS 6.22 have been successfully tested. The computer requires any video card capable of displaying text, 4 MB of RAM, a 3.5 inch floppy drive, and a hard disk or electronic flash disk. Depending on the configuration, the computer will need at least a few bus slots. At this writing (2001) it might be necessary to use a passive backplace design with a CPU card, since motherboards with more than two ISA bus slots are difficult to find.

The computer should be used with a rack mount keyboard and monitor (VGA). Some industrial computers have a VGA monitor (CRT or LCD display) and/or a keyboard built-in. These should work fine.

- Monitor Reference Clock Board (built by NIST, installed inside computer)

- Monitor Eavesdrop Box (built by NIST)

- Ethernet Board

This is optional, but it allows the monitor to upload the daily call logs to a web-based server where they can be displayed in tabular form. Any Ethernet board that includes drivers for DOS should work. More details are provided in Chapter 5.

Installing the Monitor

The monitor must be installed in a location accessible to a telephone line, to the server's serial port cables, to an on-time 1 pps signal, and to a 5 or 10 MHz reference frequency. It should also be near an Internet connection if an Ethernet board has been installed. Install the monitor as follows:

- Mount the monitor computer in an equipment rack. Situate the equipment so that the video display is located at a convenient viewing height, and so that the keyboard is at a convenient typing height. The top of the computer chassis makes a convenient resting place for the monitor's modem, so leave a gap between the bottom of the monitor shelf and the top of the computer chassis.

- Connect a power cord to the computer chassis, but leave the computer OFF for now.

- Connect the VGA monitor interface cable to the 15-pin video jack on the back of the computer chassis. Plug the monitor's power cord into an AC outlet. Turn the monitor ON.

- Plug the keyboard into the round keyboard jack on the back of the computer chassis.

- Connect a serial cable from the 9-pin serial port labeled MODEM to the external modem. The modem can be placed on top of the computer chassis. Connect the modem to the AC power, and to a telephone line using the supplied phone cables. Turn the modem ON.

- Connect your 1 pps time reference to the BNC connector on the back of the computer chassis (top connector).

- Connect a 5 or 10 MHz frequency reference to the BNC connector on the back of the computer chassis (bottom connector). The 5 or 10 MHz selection is made with the jumpers on the monitor reference clock board.

- Connect a serial cable from the jack labeled BOX on the monitor computer (9-pin jack) to the monitor eavesdrop box.

- Connect the network cable to the Ethernet card (if present).

- If you use the alarm, connect the alarm lines (banana jack) to your alarm source. If you use the alarm, the switch must be in the ENABLED position.

- Turn the computer on. The software has been pre-installed. When you see the C:\ prompt on the display, proceed to the next section.

Starting the Monitor

Start the ACTS monitor as follows:

- Type the following and press <ENTER> to switch to the ACTSM directory:

CD/ACTSM2K

- Type SETCLK2K and press <ENTER>. You have two options (select one to set the monitor clock):

  (1) call ACTS to set the computer time and synchronize the 1 pps

  (2) synch the 1 pps to the current computer clock time without calling ACTS

- Edit the monitor parameters by typing SETUPMON and press <ENTER> (see Chapter 5 for a description of the ACTSMON.ACT file).

- Start the monitor by typing ACTSM2K6 or ACTSM2KZ (the name of the executable file might vary). Then press M to start to monitor all enabled servers. The other commands you can use are executed by pressing the "hot keys" located in the lower right corner of the screen. The function of each hot key is described below (note that T is not listed on the screen display):

  M Start monitoring all enabled ACTS servers immediately. The monitor will spend the first half of the period specified in the file ACTSMON.DAT (see Chapter 5) attempting to eavesdrop on the server's serial lines. If no action occurs in the first half of this period, the monitor will call the server by telephone. The monitoring can be interrupted by pressing the ESC key every 30 s during the eavesdrop and between two successive calls. There is a message in the monitor operation status window to prompt you.

I Inspect a specified ACTS server. The program will spend 30 seconds attempting to eavesdrop on the first line of the server. If no action is detected, the monitor will call the server by telephone.

V View the contents of the configuration file ACTSMON.DAT on screen.

D Go to DOS Shell.

R Reset alarm.

T Check statistics (number of phone calls per bank and line).

E Exit the monitor program.

```
                    NIST ACTS Monitoring Window

                              (ACTSM)

 GEN:    1a  ( # OTM =  7 )          4a  ( # OTM =  7 )
 Code:   50657 97-07-28 14:55:43     50657 97-07-28 14:57:32
 Stamp:  50657 97-07-28 14:55:42.954 50657 97-07-28 14:57:31.951
 Diff:       0  0- 0- 0  0: 0: 0. 46     0  0- 0- 0  0: 0: 0. 49

 GEN:    2a  ( # OTM =  7 )          Monitor Operation Status
 Code:   50657 97-07-28 14:56:19
 Stamp:  50657 97-07-28 14:56:18.950
 Diff:       0  0- 0- 0  0: 0: 0. 50

 GEN:    3a  ( # OTM =  7 )
 Code:   50657 97-07-28 14:56:56
 Stamp:  50657 97-07-28 14:56:55.951
 Diff:       0  0- 0- 0  0: 0: 0. 49

 REF_CLK:  50657 97-07-28 14:57:48.898     PC CLK:  1997-07-28 14:57:48

Status: Idle                          Monitor ACTS/DosShell/Exit
        next call:  50657  14: 59: 47  Inspect ACTS/View Setup/Reset Alarm
```

Figure 2. The ACTS Monitor Display

If an alarm is indicated by the monitor you will hear a succession of beeps and a red ALARM message will appear on the screen below the words “Monitor Operation Status”. A file containing the time code stamp and time error will be recorded in the file ALARM.ACT. The

two banana jacks on the back of the monitor will generate a physical alarm by creating an open circuit across the two connectors if the alarm switch has been enabled on the back of the computer. The system will continue to call and will give an alarm until the bank has been resynchronized with the correct time. The alarm can be reset by typing R at the monitor keyboard, but the monitor will continue to generate alarms until the correct time is set.

Using the Monitor

Once the monitor is operational, there are only a few things you need to do:

- Periodically check the statistics on each bank to make sure that each modem is accepting calls. The statistics indicate the number of calls that each modem has taken over the past 24 hour period, with the statistics being recorded every hour. If no calls are recorded on a particular line, the modem is not responding. Fix this problem by turning the modem off, then back on.

- Watch for time code errors and errors in the offset from the reference clock (should normally be less than than 10 ms).

- If you do get a time code error, busy out the phone lines by plugging the supplied line terminators into each phone jack. Find the problem before restarting the ACTS server. Before plugging the phone lines back in, make sure that the ACTS bank is set to the correct time.

- Periodically inspect the screen for blinking red messages (these indicate error conditions). Appendix D lists the descriptions of the messages displayed in the Monitor Operation Status window.

- Check the error message files periodically by typing D to go to the DOS shell and then typing **dir *.act**. You can view the contents of any file by using the DOS TYPE command or a text editor. The contents of each file are explained in Chapter 5. When you finish, type EXIT at the DOS prompt to return to the monitor.

If you need to change the monitor parameters you will need to exit the monitor program, run SETUPMON.EXE and then restart ACTSM2K6 or ACTSM2KZ.

Chapter 4

Technical Reference for ACTS Server

This chapter provides a technical description of the ACTS Time Server, including a detailed discussion of the hardware and software.

Hardware Requirements and Configuration

The following sections describe the hardware used by the ACTS server:

- *Computer*

  The ACTS software run on a standard PC. It has been successfully tested on a number of computers using both 486 and Pentium processors, and running either MS-DOS 6.22 or ROMDOS 6.22.

- *Monitor*

  A VGA color monitor is assumed, but this may be changed to a monochrome monitor using a configuration parameter in the file MACHINE.H. The internal display driver should function in a monochrome environment with almost no other changes.

- *Disks*

  The software does not write to disk, and the required disk capacity is just a few megabytes. A read-only electronic flash disk can be used in place of a hard disk.

- *Clock Board*

  The software uses an external clock as a time reference. The current version uses the GT401 Event Timing and Controller Board manufactured by Guide Technology, but equivalent hardware from other manufacturers could also be

used. The Guide Technology board contains a free-running crystal oscillator with a typical frequency offset of ±5 x 10^{-6} per day (0.5 s per day). The oscillator can be calibrated using an external standard and its frequency aging is on the order of 0.02 s/day/month. It is therefore almost good enough to use with only occasional calibrations. The clock board can be synchronized using an external 1 pps signal, and the long-term accuracy of the time transmitted by the server cannot be maintained without such an external signal. This capability is enabled using the subroutine library functions supplied with the board, and the board is locked to an external 1 pps signal.

The address of the board can be set to any value that does not conflict with another device. The instruction manual supplied with the board explains how to do this. The default value is set at 100h. Once the hardware has been configured, the address must be added to the configuration file for the ACTS software which is named MACHINE.H. The software does not use interrupts, and the IRQ line from the board may be disconnected or vectored to any unused IRQ using the jumper on the board.

The board performs two distinct functions:

⇒ It keeps the internal DOS clock fast with respect to UTC by about 0.77 s (14 ±1 DOS-clock ticks). The DOS clock is steered by ±1 tick as necessary to maintain this relationship.

⇒ It determines the transmission time for the On-Time Marker (OTM). The software uses a fixed advance of 45 ms if the user does not echo the OTM character, and will adjust the advance using the actual measured delay if the user does echo this character. The advance is driven from the millisecond value of the Guide board in both cases. As described below, the time code transmitted to a user is computed from the internal DOS clock and is therefore transmitted to the user starting about 0.8 s early. The OTM is transmitted when the Guide board is set to 955 ms in fixed advance mode. The variable advance value is determined dynamically for each connection.

- *Serial Ports*

The software controls up to six serial ports. Each port must have its own unique base address and IRQ. Each port is controlled using eight consecutive addresses so that a port whose base address is XX0 has registers at addresses XX0 through XX7

and one at XX8 has registers at XX8 through XXF. The function of each register must conform to the standard PC architecture.

Versions of ACTS exist that support either four or six serial ports. The port parameters for the four port version are:

Port Number	**Address**	**IRQ**
1	280	4
2	288	3
3	290	10
4	298	11

The port parameters for the six port version are listed in the table below. Note that ports one through four of the six-port version are identical to the standard DOS assignments for ports COM1 through COM4.

Port Number	**Address**	**IRQ**
1	3F8	4
2	2F8	3
3	3E8	10
4	2E8	11
5	2D8	12
6	2C8	11

- *Modems*

  The ACTS server is designed to operate with up to six external modems. Modems that meet these requirements should work:

  ⇒ The modems must communicate at all speeds from 300 to 9600 bits per second using all standard formats and protocols. The actual communication speed and format is negotiated between the local and remote modems during the initial connection phase, and faster modems (14.4 kilobits per second and above) drop back automatically to 9600 bits per second (or to a negotiated slower speed if the line condition is poor).

  ⇒ The communication link between the serial port and computer must function at a fixed speed independent of the actual negotiated transmission speed for a phone call. With the modems we have tested, the communication link to the computer runs at a constant speed of 19.2 kilobits per second.

  ⇒ The modem should provide buffering to deal with the difference between its input and output speeds in a way that is transparent to the software. The worst-case condition for all speeds is when the help message is being transmitted to a user. If the speed of the connection is 1200 bits per second or faster, the modem is sent characters in blocks of 110 characters or less. The modem will receive the block of characters in about 18 ms (at a speed of 19.2 kilobits per second) and will have to hold most of these characters until they can be transmitted, which will take almost a full second at 1200 bits per second. The blocks are shortened to 28 characters each if a user connects at 300 bits per second. Again, the time required to transmit the block of text is almost one second. The average software speed is thus limited to 28 characters per second at a connection speed of 300 bits per second and 110 characters per second at all higher speeds. These limits guarantee that there will never be a long-term backlog even when the help message is transmitted to a user connected at the lowest supported speed. The average transmission rate for the time code is smaller, about 42 characters every 2 s at 300 bits per second and 56 characters per second at all higher speeds, so that the help message length requirements drive the design in all cases.

  ⇒ The modems must respond to Hayes command sequences and to signals sent via the Request to Send and Clear to Send (RTS/CTS) lines. The modems must hang-up if Data Terminal Ready (DTR) drops and not answer a call as long as it remains off. The modems must automatically initialize themselves when DTR is reasserted.

The proper position for each switch on the U.S. Robotics 14.4 Sportster modem is shown below, along with the settings stored in the modem's nonvolatile memory (this configuration will vary slightly when other modems are used).

Switch Number	Position	Function
1	up/off	Control via Data Terminal Ready enabled
2	up/off	Word result codes
3	down/on	Result codes enabled
4	up/off	Echo keyboard commands
5	up/off	Auto answer enabled
6	up/off	Carrier Detect follows carrier
7	up/off	Load Y1 configuration on reset
8	down/on	Enable AT command mode

Parameters stored in nonvolatile memory

Standard Parameters:

B0 Answer in US or ITU/T Mode
E1 Echo keyboard commands
F1 Echo off in data mode
M0 Loudspeaker off at all times
Q0 Display result codes
V1 Result codes are words
X4 Enable all result codes
Y0 Reset to state &F1

Serial Line parameters:

BAUD=19200 PARITY=N WORD LENGTH=8

Default dialing state: DIAL=PULSE ON HOOK

Additional Parameters:

&A3 Add protocol indicators to response code
&B1 Serial port operates at fixed speed
&C1 Carrier Detect enabled
&D2 Data Terminal Ready enabled
&G0 No Guard Tone for US systems
&H1 Hardware Transmit Flow Control via CTS
&I0 Software Flow Control disabled
&K1 Data compression automatically determined
&M4 Normal error control
&N0 Telephone line rate is variable
&P0 US pulse dialing ratio
&R2 Hardware receive Flow Control via RTS
&S0 Data Set Ready always on
&T5 Remote Loop-back disabled
&Y1 Disconnect on break

Contents of S Registers (most of these are factory defaults)

S00=001 Answer on first ring
S01=000 Counts incoming rings
S02=043 Escape character = "+"
S03=013 Carriage Return character
S04=010 Line feed character
S05=008 Backspace character
S06=002 Delay in seconds before dialing
S07=060 Wait for carrier after dialing
S08=002 Sets duration (in seconds) for pause option in dial command
S09=006 Delay for carrier detect in units of 0.1 s
S10=007 Delay before hanging up on loss of carrier in units of 0.1 s
S11=070 Duration of tone dialing in milliseconds
S12=050 Duration between "+" characters in units of 0.02 s
S13=001 Reset when DTR drops
S14=000 Return to command mode on "+++"
S15=000 Bit-mapped, options not used

S16=000	Bit-mapped, options not used
S17=000	Reserved
S18=000	Disable loop-back timer
S19=000	Disable inactivity disconnect
S20=000	Reserved
S21=010	Set break to 100 ms
S22=017	XON character
S23=019	XOFF character
S24=000	Reserved
S25=005	Minimum time for DTR drop to be recognized = .05 s
S26=000	Reserved
S27=000	Bit-mapped, options not used
S28=008	Connection training time = 0.8 s
S29=020	Reserved
S30=000	Reserved
S31=000	Reserved
S32=000	Reserved
S33=000	Reserved
S34=007	Disable V.32bis
S35=000	Reserved
S36=014	Reserved
S37=000	Reserved
S38=000	Hang up immediately on DTR loss

- *The Network Interface*

If the network verify option is selected, the software periodically compares its time to the NIST Internet time servers using UDP/IP time packets as described later in this chapter. The interface to the Internet is an Ethernet card. The interface requires a special driver program that is usually supplied by the vendor and is unique to the hardware. The interface card and driver must use an IRQ and address that does not conflict with other boards installed inside the server.

Software Requirements and Configuration

The ACTS time server software controls up to six dial-in modems through standard serial ports. The software provides full modem control and line supervision and operates at any speed supported by the modems, including 300 bits per second.

The software consists of a main program named ACTS.C and a number of utility subroutines that are described in this chapter. Low-level support for the serial ports is provided by a commercial library of functions. In addition, if network verify is used, the software depends on the Waterloo TCP libraries for generating UDP/IP packets and for controlling the network interface. These libraries are available in the public domain.

The software is not designed to operate in a multi-tasking environment and may not work properly in the "DOS Window" of another operating system. In particular, the software must be able to write directly to the VGA screen memory and to control the lines of the serial ports.

- *Operating System*

  The software is designed to run in a standard PC using MS-DOS or ROMDOS. The software requires support services provided by the standard BIOS and DOS interrupt routines. Although version 6.22 of DOS has been successfully tested, all services used are common to most recent versions of MS-DOS, and no particular version is assumed.

- *Compiler*

  The software was compiled using Borland Turbo C++, but none of the extended features of C++ are used, and other C compilers should work with only moderate changes. The software is built in the small memory model, and does not require extended or expanded memory. If the network verify option is used, the Ethernet packet driver will operate more efficiently if it has several buffers located above the usual 640K memory limit, but these buffers can also be located below 640K. The text segment of the software is almost 64 kilobytes long, which is the limit for the small memory model. The software could be expanded by removing unused functions from the clock-board driver software.

- *Serial Port Driver*

  Input, output, and control of the serial ports are managed by a library of subroutines that are incorporated into the software during the linking process. These routines provide support for interrupt-driven I/O. The software also controls the Data Terminal Ready line using direct input and output commands to the appropriate control registers of each of the serial lines.

- *The Network Verify Option*

  The network verify option checks whether the server time is correct to the nearest second. It does this by transmitting the same time message that is sent to clients to a verify server, which then replies with the difference between the time in the message and the time of the server. The verify service has an advantage over other methods (such as asking another machine for the time directly) in that only the transmission time of the local query is critical; the response is a time difference and can be read when it is convenient to do so. This is important because if the response were a time string it would have to be read as quickly as possible and it would be difficult to do this without locking up the ACTS software in a tight loop.

  The verify servers listen on UDP port 1123; they will accept all connections on this port. The server expects a time string of 24 characters in the same form as is used to transmit the time to an ACTS client:

  MMMMM YY-MM-DY HH:MM:SS . . .

  where MMMMM is the Modified Julian Day and the remaining characters are the UTC date and time. The server computes the time difference using the Julian Day and the time; the date must be present but is ignored since it is used to compute the MJD and will therefore be correct if the MJD is okay. The daylight saving time flag need not be checked since it is computed from the MJD and will be correct if the MJD is okay. If the server is healthy and if the magnitude of the time difference is less than 900 ms, the server responds with four ASCII characters giving the signed time difference (client – server). If the magnitude of the difference exceeds 900 ms, the server responds with +/– 900 as appropriate. If the server is not healthy or if there is a format error in the message, the server responds with +999 or +998, respectively. Thus the response is always four ASCII characters. The usefulness of the verify method depends on the fact that the network delay between the acts server and the verify server is substantially less than 1 second; this will be true between Boulder and most points in the continental United States that are directly connected to the Internet. The actual delay between a given location and the verify servers in Boulder can be measured using a utility such as "ping" or "tping". The verify service should not be used if either the measured delay or its standard deviation are more than 0.5 s.

The network verify option requires a socket interface to the Internet. The interface is implemented using the driver supplied with the network card together with the packet driver and socket library provided by the Waterloo TCP package. This code is in the public domain. Other drivers and network cards could be substituted provided that the resulting combination supports the standard packet-driver interface to the Internet. Likewise, other TCP packages would also work if they provide libraries to use the standard socket-type of network calls.

- *Driver for the Clock Board*

  If the Guide Technology GT401 clock board hardware is used in the configuration, then its driver software must be linked with the ACTS software. This driver software is supplied by Guide Technology with the board. The driver routines have been combined into a library using the Microsoft C compiler set to the Large Model, and this library may not be compatible with the Borland Turbo-C system that is used to compile the remainder of the system. In particular, the ACTS software does not need Large Model routines and the Borland system does not recognize the library routines INP and OUTP by default. These routines can be defined by including the file <DOS.H> into the driver code.

  The time of the clock board is locked to the integer second using an external 1 pps pulse; this phase lock is enabled using software (SETGT.EXE) calls to the driver.

Software Operating Principles

This section describes how the ACTS software operates, beginning with the initialization procedure.

Initialization

The start-up code is configures the serial ports and the modems. Each serial port is configured to operate at a fixed speed of 19200 bits per second with no parity, 8 data bits and 1 stop bit. It is also possible to configure the software so that it communicates at a fixed speed of 1200 bits per second. This slower speed could be

used to control older 300/1200 baud modems. The modems automatically buffer each message if a user connects at any lower speed down to 300 bits per second. The communications protocol will not accept XON/XOFF flow control in either direction; the modems and the serial ports communicate status information using Data Terminal Ready (DTR), Clear to Send (CTS) and Request to Send (RTS).

Each port has dedicated input and output buffers, each of which is 132 characters long. Although there is very little input, the length of the input buffer is defined so that it is larger than the largest response of the modem, which is a connection message in verbose mode. The full input buffer might also be needed if a user echoes all characters, and not just the OTM, back to the server. No more than 120 characters of the output buffer are normally used; the extra characters are reserved.

If the software configures the port successfully, each modem is tested by sending it the reset command ATZ followed by a carriage return. The program expects a response of "OK" and will print an error message if it is not received. In addition to configuring the modems, this portion of the code initializes the clock board and its driver if this option is present and the network interface if the verify option is selected.

When the modems have been tested and the initialization of the other devices is complete, the software enters the steady-state operating mode. The steady-state operation of the server is driven by two parameters: the system time and a state vector giving the status of each of the serial ports as an integer.

The System Clock

The clock oscillator on a standard IBM-compatible PC runs at a frequency of 1.19318 MHz. This frequency is divided by 65536, and the resulting frequency generates an interrupt on every cycle. The interrupt frequency is thus 1193180/65536 = 18.206 ticks per second. The processor increments a 32-bit register on each interrupt and resets the register to 0 at 00:00:00 every day. The register therefore keeps time in units of the tick period, which is 1/18.206 s or about 55 ms. Converting the internal time in ticks to hours, minutes, and seconds is the responsibility of the operating system. This conversion is complicated by the fact that there is not an integral number of ticks per second so that most exact second times do not exist. The time 00:00:01, for example must be found by interpolation between the 18th tick after midnight, which is time 00:00:00.98 and the 19th tick after midnight which is time 00:00:01.04. This is a serious issue for the ACTS

system since it transmits the time in the usual way with an exact integer value for the seconds. If the system clock were used for the time reference without some form of correction or interpolation, the time at which a message was transmitted would vary by up to 1 tick (55 ms). As an additional complication, the rate of the clock oscillator is not accurate or stable, and the time of a typical PC may drift several seconds per day or more, which is on the order of milliseconds per minute. This frequency aging has a deterministic component driven by the temperature, but there are other factors that are not constant or predictable. Thus, although it would be simple to use the PC clock as the time reference, it is simply not up to the task.

We have addressed these problems using a two-tiered system. The first tier uses the PC clock to define the integer second only. It is set roughly 0.77 s (14 ±1 ticks) fast and this relationship is maintained using an external reference as described below. The PC time is split into two quantities: the date and time to the integer second and the fraction of a second expressed as a number of ticks ranging from 0 to 18 (or 19). The time to the integer second is used to construct the time code and the tick value is used to drive the state of the system as described in more detail below. The second tier uses an external hardware pulse to specify the actual arrival of the second. This pulse is used to transmit the OTM and to keep the PC clock synchronized as described below.

The Steady-State Cycle

This cycle is implemented in program ACTS.C. It is driven by the tick value described above. This is an integer giving the time in ticks since the last integer second and ranges from 0 to 18 or 19.

The cycle starts as soon as possible after the 0th tick. The program calls subroutine ***timcod()*** to generate the time message using the current system time. This subroutine in turn calls other subroutines to calculate the Modified Julian Day (MJD) number, the daylight saving time flag, etc. The time code is stored in global character array *obuf* and displayed on line 15 of the screen. A shorter version of the time code is also generated and stored in global array *obuf3*. This time code does not have the MJD or UT1 parameters. In addition, its value in seconds is one more than the "standard" time code stored in *obuf*. The shorter code is sent to users who connect at 300 bits per second. Note that the shorter time code is generated by blindly adding one to the seconds value of the standard time code without considering the possibility that this increment may cause the time to overflow at the end of a minute. This point is important in the subsequent discussion.

The ***timcod()*** subroutine also manages the leap second flags, and the current value of these flags is used to update the screen display. If the VERIFY and Guide board clock are enabled, the screen displays for both of these systems are updated at this time. The time to the next comparison is decremented in both cases, and the new value is displayed on the screen. This portion of the code takes about 90 μs to complete, so that the tick value is almost certainly unchanged.

The next portion of the code loops through all of the serial ports. This portion begins when the time code is ready and the housekeeping is done. The action depends on the state value for each port. The state values are integers and are maintained independently for each port. The action in the loop starts with the previous value of the state for that port and continues until a branch indicating "end of action" is reached (all state values start at 0). The operation continues with the next port, and when all ports have been tested the next phase of the loop begins. This loop over all serial ports is not executed again until the next second. The table on the following pages lists the state values and their associated actions.

State	Action
0	Port is idle, looking for "R" of "RING" Special action for State 0: ⇒ If both digits of the seconds are zero (i.e., the time is xx:xx:00), then set state 82, which will force a modem reset on the next second and reset the *icol* parameter for this port (see below) and end of action. If either digit of the seconds is not zero then continue with the common action.
1	"R" of ring has been received, look for "I"
2	"RI" of ring has been received, look for "N"
3	"RIN" of ring has been received, look for "G" Common Actions for states 0 through 3: ⇒ Test to see if a character has been received on that port since the last cycle. If nothing has been received and if *icol* for this port is zero then no state change and end of action. ⇒ If nothing has been received and if *icol* is non zero then increment *icol*. If it is >8, then set state 82 and end of action. Otherwise end of action. This acts as a timeout counter which starts when any letter of RING is received and forces a reset if the remaining characters are not received within 8 loops. ⇒ If a character has been received compare it to the character expected. If received character matches expected, advance state, set *icol* for this port to 1 and continue immediately. Once this case sets *icol* to 1, the remaining characters of "RING" must be received before *icol* overflows to 8, since if that happens the test above will force a reset. This is to prevent a spurious character that happens to be one of RING from locking up the port waiting for the remainder of RING which will never come. If another character is received then ignore it and end of action.

State	Action
4	"RING" has been received and modem is answering Action: ⇒ Change port status on screen to "Ring" ⇒ Advance to State 5 and end of action.
5 to 30	Waiting for "CONNECT..." or "NO ANSWER ..." from modem Action: Wait for “C” or “N” character as first character in modem message. ⇒ If no character is received, increment state and end of action. ⇒ If a “N” is received, set state to 82 and end of action (modem is sending NO CARRIER, hang up). ⇒ If a “C” is received, set *icol* for this port to 0 and continue immediately. *icol* will function as a time-out counter (modem is sending CONNECT). ⇒ If character is not "C" or "N", set state to 5 and end of action.
31	Fall through while waiting for "CONNECT", time-out We will reach this state by an increment from state 30, which means that we are still waiting for either the C of CONNECT or the N of NO CARRIER and we have given up waiting. Set state to 82 and end of action. This will force a hangup/modem reset on the next second.
32	"C" of CONNECT has been received, look for "O".

State	Action
33	"CO" received, look for "N".
34	"CON" received, look for "N".
35	"CONN" received, look for "E".
36	"CONNE" received, look for "C".
37	"CONNEC" received, look for "T". States 32 through 37 all test to see whether a character was received: ⇒ If no character received, increment *icol*, and set state to 82 if *icol* > 3 and end of action. This forces a hangup/reset on the next second if the CONNECT message does not complete in 3 loops. If *icol* < 3 then no change of state and end of action. ⇒ If character received, check against character expected. ⇒ If expected character found, advance to next state. If other character found, ignore it, do not advance state and continue immediately. Note that computer talks to modem at 19,200 bits per second so entire message should be read in one or two cycles.
38	Complete CONNECT message received If the server accepts connections at 300 bits per second we now must read the remainder of the connection message and check the input buffer: ⇒ If buffer is empty then increment *icol* for this port and set state to 82 if *icol* > 5 and end of action. This forces a reset/hangup on the next second if the complete connection message is not received in 5 loops. If *icol* < 5 then end of action immediately. ⇒ If buffer contains a digit character, then the speed of the connection is at least 1200 bits per second. Set the *lsp* flag for this port to 0 (not low speed) and advance to the Action-2 portion of this state.

State	Action
38 (cont.)	⇒ If the buffer contains a line feed character, then the modem is sending just the word CONNECT with no speed, which means the speed is 300 bits per second. Set the *lsp* flag for this port to 1. ⇒ If the input buffer contains any other character, then ignore it, read the next one and continue immediately. Before exiting this state, print connect message on screen, display the speed with the H- (1200 bits per second or faster) or L- (300 bits per second) prefix, then set internal variables to get ready to send time code.
39	Increment call counter for port. ⇒ If speed is 1200 bits per second or higher, increment call counter and advance to state 40. ⇒ If speed is 300 bits per second, delay for 1 s and advance to state 40.
40	Send Welcome Message ⇒ Empty input buffer of any characters remaining from connect message. ⇒ Copy welcome message to output buffer until terminating NULL is encountered. This will start transmitting the welcome message. Move up to 25 characters if the connection speed is 300 bits per second and up to 110 characters to the output buffer at higher speeds. If the transfer ends or the output buffer fills before NULL encountered, save pointer, do not change state and end of action (the remainder of welcome message will be transmitted on next cycle). The message is copied from the memory buffer without alteration except that if the "#" character is encountered in the message it is replaced by the number of the port being addressed. ⇒ Advance to state 41 when NULL is read, end of action. Note that transmission of output buffer continues during the remainder of this second.

State	Action
41	Transmit column headings, wait 1 s to be sure message is finished. ⇒ Copy column-headings from memory buffer to output buffer. There are two headings, one for the shorter message sent at 300 bits per second and the other for the standard message sent at all other speeds. Copy the appropriate message to the output buffer one character at a time until the terminating NULL is found or until the output buffer is full. Transfer no more than 25 characters to the output buffer is the connection is at 300 bits per second, and end action with no change of state if the output buffer fills or if the 25th character is reached. End action and advance to state 42 when the terminating NULL of the message is encountered. ⇒ The portion of the welcome message left in the output buffer when state 41 started can not be more than 25 characters at 300 bits per second or 132 characters at any other speed so that it is guaranteed to finish before we get to state 42 on the next second.
42-81	States 42 through 81: Send time messages If this is state 42, and if the connection for this port is at 300 bits per second and the least significant digit of the short time code is even, then advance state and end action immediately. The time code is only sent on odd seconds for slow connections. Common Action for all other states: 1. If input buffer contains a character, then read it. ⇒ If it is "?" set state to 100, initialize help message pointer and end of action (prepare to respond to help message request). ⇒ If it is "%" set state to 82 and end of action (hang up request). ⇒ If it is "~" and if speed is greater than 300 bits per second, then set state 103 and end of action (get ready to send statistics). Set *icol* parameter for this line to 0; it will be used as index into statistics arrays.

State	Action
42-81 (cont.)	⇒ If the character is an OTM then it is much too late, since the path delay might be very long. Set maximum possible advance and continue. ⇒ Ignore all other characters. If the input buffer contains more than one character, then keep reading until the buffer is empty, testing each of the characters as outlined above. This loop purges the input buffer of any characters that may have been echoed including the time code itself. 2. If modem status shows no CARRIER DETECT, set state 82 and end action (user hung up). 3. If measured-delay mode has been incorporated into the software, check to see whether an echo of the previously transmitted on-time marker was received and processed by the delay-measuring code. The j-th bit is set in the echflg word when the OTM is sent and is reset when its echo is received, so that no echo was received if the bit is still set. If an OTM echo was not received then reset both the OTM and the advance to their default values in this case. This means that either an echo was not received within the time window or its delay was not consistent with the previous values (see below). 4a. If this connection is operating at any speed faster than 300 bits per second, send return, new line, and time message to this port. Copy text to output buffer and begin output transmission. If this port is operating in measured-delay mode, the delay value in the time-code is modified just before transmission by inserting the actual delay for this port into the message instead of the nominal delay of 45 ms which was built into the message by the ***timcod()*** subroutine. Note that the transmission consists of 51 characters. The transfer to the modem takes place at 19200 bits per second. The modem retransmits the message to the user, and this retransmission will take 425 ms at 1200 bits per second, and less time at faster speeds. Since the local clock is 14 ticks fast, there is a delay of about 340 ms from the end of the time message to the actual time, therefore 340 ms is the maximum OTM advance at 1200 bits per second. This maximum advance would mean that the OTM would be sent immediately after the message text finished. This maximum advance would be larger at faster transmission speeds.

State	Action
42-81 (cont.)	4b. If the connection is at 300 bits per second, then it gets the shorter time code every other second. Examine the seconds value of the shorter time code in *obuf3* and see if it is odd. If it is odd, then begin transmitting. Note that this code was generated with a seconds value one more than the standard high-speed code, so that users connected via faster ports are getting a code with an even second that is one second less. Note that if the seconds value is odd, then it cannot be less than 1 and that the actual time code would have a seconds value of 0 in that case. Likewise, the maximum odd value would be 59, with the faster message having a seconds value of 58. In both cases, the remainder of the date and time must be the same, both codes will always have the same year, month, day, and so on. If the time code is odd and is transmitted, then END ACTION immediately, do not continue below. If the second in *obuf3* is even, then do not send the time code and continue below. Note that the transmission consists of 41 characters and will take 41/30 seconds. The message will therefore end about 300 ms into the next second. 5. If NOHANGUP status is not set (*nohup* = 0), then advance state and end of action. If NOHANGUP status is asserted (*nohup*=1), then do not advance state, and transmit time messages until client breaks the connection. 6. Set the bit in the *echflg* parameter that corresponds to this port showing that it was sent a time code and that it should therefore be sent an OTM below. A copy of this flag is also used to show that an echo of the OTM should be expected from this port and its arrival time should be recorded if measured delay mode is implemented. End of action. If this connection is at 300 bits per second, then paragraph 6 is executed on the even second in *obuf3*, so that the time code is sent when its seconds value is odd and the OTM is sent at the appropriate place in the following even second. If this connection is running at any higher speed, then the time code and OTM are sent every second. The same thing is true for paragraph 5, it is executed every second at 1200 bits per second and above but only every other second at 300 bits per second. Thus, the number of messages transmitted is the same at both speeds, but the maximum connection length at 300 bits per second is twice as long as at any of the higher speeds.

State	Action
82	End of time messages ⇒ Turn off Data Terminal Ready (modem will hang up). ⇒ Show Reset status on screen for this line. Advance to state 83 and end action (delay 1 s and then re-initialize modem).
83	Prepare for next call on this line ⇒ Turn on Data Terminal Ready (modem will answer next ring). Check Data Set Ready and Clear to Send lines from modem. Show "Modem ok" if asserted. Otherwise set display to "No Modem". ⇒ Advance to state 0 and end action (return to idle state and wait for next call).
90	Do-nothing state if port disabled
100	Transmit help messages 1. If carrier detect is lost or if "%" character received on this port then advance to state 82 (hang up on next second) and end action. Otherwise continue: 2. Copy up to 28 characters at 300 bits per second or 110 characters at all higher speeds from help message at current position of help message pointer to output buffer and start sending the characters. If terminating NULL not reached, advance help message pointer and end of action. If terminating NULL encountered, then advance to state 101 and end action. This action transfers no more characters to the output buffer than can be transmitted to the user in 1 s and it is guaranteed to complete in 1 s at all speeds. There is therefore no need for flow control or other delays. The help message is transmitted in 28-character chunks to a 300 bit per second user and 110-character chunks to all other users until it has all been sent. If fewer than 20 characters are copied to the output buffer on any cycle, then assume port is hung. Advance to state 101 unconditionally even if help message is not completed.

State	Action
101	Waiting for help message output to finish. ⇒ Advance to state 102 and end of action.
102	Help message is finished; hang up on next second. ⇒ Advance to state 82 and end of action.
103	"~" character received. Get ready for statistics. ⇒ Advance to State 104. If input buffer is full, continue immediately with state 104. Otherwise end of action, thus delay for one cycle.
104	Parse remainder of send statistics request. ⇒ If input buffer is empty or character is not "s", then set state 82 (hang up on next second) and end of action. The leading "~" is assumed to be a glitch. ⇒ If character in input buffer is "s" advance to state 105 and end of action.
105	Send Statistics. ⇒ If carrier detect is lost, advance to state 82 (hang up on next second) and end action. ⇒ *icol* parameter for this line is pointer into statistics arrays. On each cycle, send 24 numbers giving number of calls for each hour for each COM port. Each value is a three digit number with one leading space so that each line is 4 × 24 or 96 characters long + <cr> + <lf> = 98 characters. This transmission is guaranteed to finish in 1 s. The first four lines give call counts for each hour of "today" for each COM port in order and the second four lines give corresponding values for "yesterday." Advance *icol* after each line is copied to output buffer. Advance to state 82 when icol reaches 8, otherwise leave state unchanged and end action. Thus, the line continues in this state until all statistics have been transmitted and then advances to hangup/reset state.

This is the end of the loop over all serial ports. The next phase of the cycle begins immediately as follows:

- The keyboard buffer is tested to see if it contains any characters. If it does, they are loaded into array *kbuf* by ***kybcmd()***. If ***kybcmd()*** returns a status > 0, then a completed command is ready to be parsed and processed. If it returns 0, then a command is not complete and this phase ends. If it returns –1 then an empty command has been entered and this phase ends.

- The software calls ***docmd()*** if a complete command has been entered. This subroutine returns 0 if the command is parsed and completed, and a non zero value if the command is parsed but not completely processed.

- If the fourth bit of the return status is non-0, the command is a COM command with bits 0 to 2 specifying the port number. If the third bit is 1, the request is to enable the port. Set the state of that port to 82 in that case. This will cycle DTR, reset the modem and eventually return to idle status. If the third bit is 0, then disable the port by turning off DTR and set the state to 90, which is a do-nothing loop.

- If the fifth bit of the return status is not zero, the command is a halt command. Close all ports and exit.

The remaining commands are processed within ***docmd()*** and return a status of 0. The other non zero status returns are reserved for expansion.

Verify Clock Phase of ACTS software

This phase is used to check the local clock against other time sources. This phase is broken into two parts, a check of the DOS clock against the Guide Technology board, and a check of the DOS clock against network time servers. Both of these checks are compiled into the code if the corresponding parameters GT and VERIFY are defined in the file MACHINE.H as described below.

If a Guide Technology clock board is present, the DOS clock time is compared to the Guide board, which is in turn synchronized to UTC using an external 1 pps signal. This comparison has a dynamic range of 24 hours and a resolution of 1 ms, but the wide dynamic range is not needed if the network VERIFY option has been selected, since that

option will normally keep the DOS clock on the correct second. The primary use of this comparison is to keep the DOS clock fast by 14 ± 1 ticks. Since both the DOS clock and the Guide board can be read with millisecond resolution, the difference can be computed at any point in the cycle, and the comparison is made at this point for convenience. If the difference between the DOS clock and the Guide board does not equal 14 ± 1 ticks, the DOS clock is steered by ±1 tick in the appropriate direction. If the difference is equal to the expected value, then no action is taken. Note that both clocks can be read with negligible delay so that the time comparison (and any necessary adjustment) can be completed immediately.

Once the DOS clock is checked against the Guide Technology board, the synchronization between the Guide board and the external 1 pps is verified. The 1 pps that synchronizes the Guide board clock is also connected to the "time-tag" input on the board. This input stores the internal board time whenever a pulse is received. Since these pulses are used to synchronize the clock, the time-tags should be exact integer seconds. Thus, the existence of the time-tags means that the 1 pps signal is present and the fact that all time tags are exact integral seconds means that the clock is synchronized to the external 1 pps. A message is displayed on the screen if either tests fails. The time tags also contain more significant digits including the minute, hour, and so on, but these are not used in this process; only the existence of the time tags and the fact that they are exact integral seconds is important. The magnitude of the time tags is reduced periodically by redefining the reference date from which they are measured. This capability is described in more detail in the instruction book for the Guide board and in the source code comments. The idea is to be sure that the time-tags can always be processed without loss of significance. The time tag is a six byte quantity in units of microseconds. Since we are interested only in the millisecond portion, we want this value to be modulo 1000. We avoid multiple-precision integer operations by noting that multiplication and modulo division commute.

If VERIFY is defined, the time of the local clock is compared against other time servers on the Internet. The parameter *vrfydue* is decremented once each second and a network verify is initiated when it reaches +1. The number of verify hosts to be used and the address of each one of them must be specified in file MACHINE.H and compiled into the software. Each verify server may be individually enabled or disabled using the /VERIFY command described below. The time code message is sent to a verify host (if that host is enabled) using UDP/IP and *vrfydue* is set to −1 to show that a time code message was sent and a

reply is expected. Since the local clock is running fast by 14 ± 1 ticks, the time code cannot be sent to the verify server until after the 15th tick, when it will actually represent the correct second. The transmission portion of the verify is therefore initiated at the very end of the loop (after the 15th tick), and the software continues immediately by returning to the top of the loop.

The read and compare function of the verify operation is performed at this point in the cycle. If *vrfdue* has been set to −1 by the transmission code (on the previous cycle) and if the current verify server is enabled, the program tests the state of the network interface to see whether a reply has been received. This test may take up to one tick to complete. If no reply is received, this portion of the verify phase ends and the test for a reply will be repeated at this point in the cycle on the next second. The expected reply is the time difference between the local clock and the server as a four digit ASCII number. When a reply is received, the reported time difference is stored and the VERIFY phase advances to the next verify server (if there is one) or resets the *vrfdue* flag to VRFINT, the number of seconds between verifies. If another server is selected, a message will be sent to it later in this cycle and the response will be tested on a subsequent cycle. If *vrfdue* is −1 but the current verify server has been disabled, then no packet was sent and no reply is expected. Set the corresponding value on the display to OFF and advance to the next server below.

When all servers have responded (if enabled) or not if explicitly skipped (if disabled), the data are examined, and the summary status is reported in parameter *nd*. If *nd* = 2000, then the servers have been disabled. If *nd* = 1500, then more than one server responded and they do not agree on the time difference between the local clock and network time. The actual values reported by each server are displayed on screen. If the magnitude of *nd* is less than 900, then all active servers agreed on a value for the time difference between the local clock and the time of the servers. This value is in units of seconds, with a positive value signifying that the local clock is fast. If either server reported a time difference of 999 or 998, then it has failed or the network connection is having problems. If both servers report 999 or 998, it probably means that the local network interface has problems.

If all active servers agree that the local clock needs an adjustment, the adjustment is made by calling ***adjtck()***. The time difference in seconds is converted to ticks by multiplying by 18, and the resulting correction is made to the local clock in one step. The local clock should now be on the correct integer second, although it might be wrong by one second in an unusual case. In either of these situations, the

remaining correction will normally be performed by the Guide clock loop, which keeps the fractional second correct. Note that because of the interaction between these two loops, the time of the local clock may be wrong by a significant fraction of a second after the network verify loop has finished and before the Guide Clock loop has begun. If the magnitude of *nd* is greater than 900, then either no server is active, no server responded, the response was in error, or more than one server was queried and they did not agree. No time adjustment is done in any of these cases. If this loop determines that a correction is necessary, a comparison is rescheduled on the next second to insure that everything is working.

As discussed above, the time message is transmitted after the 15th tick, when the time in the current time message should be correct to the nearest second. The transmission delay is generally about 50 ms so that the response of the time server ought to be 0 in almost all cases if the local clock is running on the correct second. The verify loop is not used to make sub-second corrections since the network delay is unknown and variable at this level.

Sending On-Time Markers (OTM) to Each Active Port

The next phase of the loop is sending the OTMs to each active port. This is done using the Guide clock board. The Guide board must be present for proper operation at the millisecond level of uncertainty.

The software initially uses a fixed advance of 45 ms. Since the Guide board is synchronized to UTC using an external 1 pps signal, the OTMs are transmitted when the millisecond field of the time read from the Guide board is 955 ms. The software enters a tight loop waiting for this event starting at the 10th tick of the DOS clock. Since the DOS clock is nominally 14 ticks fast, the time on the Guide board is actually about 750 ms in the previous second, so that this loop is executed for about 250 ms, which sets the maximum value the advance could have. The advance for each port is specified in the corresponding entry in the *adv* array, and the OTM character (either "*" or "#") is specified in the OTM array. A bit is set in the echflg parameter for each port that was sent a time-string earlier in the program, and each of these ports is sent its OTM when its advance value is read from the Guide board clock. The OTM of the display is assigned a 40 ms advance, which makes the displayed OTM look on-time. The corresponding bit of the *echflg* parameter is cleared as each OTM is sent, and the operation is completed when the *echflag* parameter becomes 0 indicating that all of them have been transmitted. Since the

advances of all ports are generally more than 40 ms, the loop usually ends when the OTM of the display is transmitted, which means that this phase of the loop usually ends when the actual time is at 960 ms of the previous second. This might not be true for all cases, some nearby connections might require an advance as small as 32 ms or even less. The actual order of advance transmissions does not matter.

There is nothing special about a 300 bit per second connection at this point. It is sent an OTM just like any other line if the corresponding bit in the *echflg* word is set. As described above, this bit will be set only every other second on the second after the one in which the time code was transmitted.

Note that the advances are driven from the ms portion of the time as read from the Guide board, the remainder of the time read from the board is not used at this point since it still indicates the previous second.

The OTM-transmission phase of the software generally ends when the time on the Guide board is no later than 960 ms of the previous second (although it might be as late as 968 ms for a local connection which might have an advance as small as 32 ms). The corresponding time on the DOS clock is somewhat later than 13 ticks in the current second. If measured delay mode has been compiled into the software, the next operation is to look for the echoed OTMs. Since the next time code is not generated until the zeroth DOS-clock tick, there are five or more ticks during which the software looks for echoed OTMs. This represents a time interval of about 275 ms, which is nearly the same duration as the maximum advance described earlier.

The software that measures the OTM delay is a nested set of tight loops. The outermost loop begins after all the OTMs have been sent (see above), and continues until the DOS clock time changes to the zeroth tick of the next second. As discussed above, this loop will be executed for about 275 ms. The next loop is a loop over all of the COM ports. The input buffer of each port that was sent an OTM and for which an OTM echo has not yet been received is examined. Characters that are not OTMs are ignored, except for "%", "~" and "?", which are processed as described in the discussion for states 42 through 81. If an OTM is received, the transmission delay is measured and the advance value may be modified as described below.

Let T be the millisecond portion of the Guide board time at which the OTM was sent and R be the corresponding time at which it was received. The initial value of T is 955 ms, as discussed above. If $R > T$, then the previous advance was too large since the millisecond value has not yet wrapped around to the next second. The one-way transit

time is (R – T) / 2, and the next transmission time should be set to be 1000 – (R – T) / 2, so that the next OTM arrives just as the Guide board clock reaches the even second.

If R < T then the millisecond value of the Guide clock has wrapped around to the next second. The round-trip delay is (1000 – T) + R so that the one-way delay is 500 + (R – T) / 2. The transmission time should be 1000 minus this value or 500 + (R – T) / 2 so that the next OTM will arrive just as the Guide board clock reaches the even second as above.

To prevent false triggering, the advance computed above is tested to see whether it is less than 30 ms and is discarded if it is. An advance of 30 ms means that the next computed transmission time is 970 ms. An advance this small will not be observed even on a local loop and is probably a mistake, possibly a human who is "holding down" the OTM key on a terminal emulating program. Likewise, an advance greater than 300 ms is discarded; the software cannot handle it and it is almost certainly an error in any case. Next, the transmission time computed from the current OTM is compared with the values for the two previous ones and the current value is used only if it agrees with both previous measurements to within ±12 ms. Thus, the software will not change the OTM advance until the third OTM, and then only if a consistent delay is received. If any of these tests fails, the OTM is reset to "*" and the advance is reset. If the measurements appear to require a very small advance, then it is possible that the true advance is very large and we are seeing an OTM from the previous second; the software tests this possibility by setting the advance to its maximum possible value; otherwise the advance is reset to its nominal value. The current advance estimate is pushed onto the stack of previously measured values so that a time step in the true delay of the connection will be accepted after the third consistent value with this new delay is received. If an echo is received, the value of R and the current OTM (* or #) are displayed on the screen whether or not the OTM or the advance are modified.

The bit that specifies that an OTM was sent to this port in the *echflg* parameter is reset to 0 once the OTM has been received whether or not its delay measurement was used to modify the advance parameter. Subsequent OTMs received on the same COM port will be ignored, so that transmitting a long-string of OTMs each second will result in the first one being processed and the remainder being ignored.

Note that 300 bit per second connections are not handled in a special way in this portion of the code. If a connection of this type was sent an OTM in this second

(which means that it was sent a time-code in the previous second) then the software will look for an echo. The advance for a connection at 300 bits per second is about the same as for any other speed so that the same considerations apply.

If VERIFY operations are supported, these are also done in this portion of the program after the 15th tick. Since the DOS clock is running 14 ticks fast, the actual time has now advanced to the point where the time code is correct to the nearest second, a VERIFY packet is sent if *vrfdue* indicates that it is needed and the currently addressed server is enabled. The software does not wait for the response to the packet that will be received and parsed during the receive portion of the next cycle as described above.

The displayed OTM is also turned off at this point, so that the displayed marker appears to "blink" for about 100 ms each second.

The software then returns to the top of the loop and the complete process starts again.

Dynamic Range of the Transmission Delay Measurements

The implementation depends on the fact that the PC clock is running fast by 14 ± 1 ticks. The time code is generated at the start of the PC second. Under worst-case conditions (the PC clock only 13 ticks fast), the time code construction begins when the actual time is no later than 286 ms into the previous second. The actual time is more likely to be about 231 ms in the previous second on the average. The transmission of the time code begins immediately. The high speed code consists of 51 characters and can take no longer than 425 ms to transmit at 1200 bits per second so that the transmission will be finished by 711 ms of the previous second at the latest. The low speed (300 bit per second) code consists of 41 characters and will therefore take about 1300 ms to transmit. It will therefore arrive no later than 586 ms into the next second and the actual time is more likely to be 531 ms into the next second. Again assuming the worst case (the PC clock time offset is 15 ticks), the time code for the next second will be constructed when the actual time is 176 ms into the second. Thus, for a high speed transmission, a interval of 465 ms (1176 – 711) exists between the latest time the transmission of one time code can finish and the earliest time that the construction of the next one must begin. The processor is idle for 99.7% of this interval, the exception being the 1 cycle out of 300 when the verify time packet is sent. The transmission at 300 bits per second finishes earlier in the

next second so that measuring the low speed delay does not drive the design requirements. This period of 465 ms can be used to measure delays up to 232 ms without using interrupts or interfering with other portions of the code. This delay is long enough for almost all paths, although it might not be long enough for a two-way path via satellite. Longer delays could possibly be measured using external hardware or if a reliable interrupt handler could be implemented to run in DOS.

Subroutines Used in the ACTS Software

The subroutines used in the ACTS software are documented in this section. These routines were written using the C programming language.

Name: adjclk

Usage: void adjclk(xdiff,cmos,ldst)
```
float xdiff;
int cmos;
int ldst;
```

This subroutine adjusts the time of the local clock by algebraically adding *xdiff* seconds to it. If *xdiff* is negative the local clock is retarded. If *cmos* = 1, then the CMOS clock is adjusted as well, and if *ldst* is 1, the daylight saving time flag of the CMOS clock is set.

The subroutine gets the local time using function 0x2c of interrupt 0x21c, applies the correction specified by *xdiff* and writes the time back using function 0x2d. The adjustment knows about the vagaries of the calendar including leap years. This process may result in losing 1 tick on the local machine if the tick changes between the time the clock is read and the time the updated value is put back.

If parameter *cmos* = 1 the system time is copied to the CMOS clock after it has been corrected. The value of the *ldst* flag is copied into the daylight saving time register of the CMOS clock.

Name: adjtck

Usage: void adjtck(jtk)
int jtk;

This subroutine adjusts the time of the DOS clock by *jtk* ticks. The DOS clock is advanced if *jtk* is positive and retarded if it is negative. The operation is performed asynchronously with respect to the clock interrupts and an interrupt may be lost if it occurs while the clock is being adjusted. This may retard the local clock by an additional tick beyond what has been requested; a requested advance of +1 ticks may result in no change to the clock and a request advance of –1 ticks may result in an actual change of –2 ticks. In addition, the operation is not performed if it would result in a carry or borrow from the low 16 bits of the time into the upper 16 bits. This is a relatively rare condition that happens only about once per hour.

Name: advdat

Usage: void advdat()

This subroutine advances the current date by 1; the month and year are also advanced if necessary. It is needed because the system date does not advance properly when a task has control of the machine at midnight. This subroutine knows about the calendar and about leap years.

If the leap second flag is set and if the advance moves the date to the last day of the month then a special internal flag is set showing that the leap second is due at the end of the current day. This special flag is checked by the ***timcod()*** subroutine to decide whether a leap second should be inserted.

If the leap second flag is set and if the advance moves the date to the first day of the month, then the leap second flag is reset to 0 since the leap second has just happened.

In addition, this subroutine copies the "current" call counters to the "previous" call counters and then sets the current call counters to 0. These counters are implemented as two two-dimensional arrays, with the first dimension of size 4 and specifying the COM port and the second of size 24 and specifying the hour.

Name: clrscn

Usage: void clrscn(mode)
int mode;

This subroutine clears the screen. If mode = 0 this subroutine clears the screen by turning off the intensity bits everywhere. The character at each position is not altered. If mode = 1 this subroutine turns on the screen by setting the intensity bits to 7 everywhere. The character at each position is not altered. If mode = 2 this subroutine clears the screen by setting every position to blank. The mode of each position is set to 7.

This subroutine will work with a VGA or monochrome display. The only difference between the two displays is in the origin of the screen memory. The type of display is retrieved from parameter VGA in the MACHINE.H file.

Name: cnvbcd

Usage: char cnvbcd(val)
int val;

This subroutine converts the integer value passed in *val* into two 4-bit BCD digits. The two digits are packed into one byte and are returned by the function. This routine is used to format the values for setting the CMOS clock. This subroutine is prepared to handle numbers from 0 to 59 which is the range needed for clock time. It will do the wrong thing without an error message if it is given a negative value or a positive value greater than 59.

Name: cvt2jd

Usage: long int cvt2jd(yr,mo,dy)
int yr;
int mo;
int dy;

This subroutine computes the Modified Julian Day (MJD) number corresponding to

the date specified in *yr*, *mo*, *dy*. It returns the MJD as the value of the function. If the value of the year is greater than 1900, it is used as is; if it is less than 1900, it is treated as 1900 + the specified value. Thus 1994 is used as specified and 94 is treated as 1994. A year of 104 will be treated as 2004. The value of mo should be in the range of 1 → 12 and *dy* should be in the range of 1 → the maximum value for that month, but these variables are not checked and no diagnostic is provided if they are out of bounds. The subroutine will function correctly if *dy* is set to the day number of the year with *mo* = 1, but will provide unpredictably wrong answers if the month is out of bounds.

The subroutine knows about leap years through 2099, but will fail in 2100 by treating that year as a leap year even though it will not be one.

Name: docmd

Usage: int docmd()

This subroutine parses a command entered from the keyboard and executes the command if possible. All commands begin with the "/" character and end with a carriage return. In general, the commands may be abbreviated to a / followed by the first letter of the command followed by any remaining parameters as needed. Letters may be entered in upper or lower case either in the full or abbreviated forms of the command. Parameters enclosed in angle brackets (such as <j >) must be replaced by a numerical value chosen from the values discussed in the text; commas and other punctuation must be used exactly as specified.

The command is not parsed until the carriage return is received, and typing errors may be corrected by erasing the mistake using the backspace key and then typing the corrected character. A command cannot be edited once the terminating carriage return has been typed.

The return value from the subroutine is 0 if the operation is completed. The return value is –1 if the command had a syntax error, and > 0 if the command requires additional processing in the main program. The return value in this case specifies what needs to be done by the caller.

This subroutine recognizes the following commands:

/COM <j>, <k>, abbreviated as /C <j>, <k>

This command executes command <k> on COM port <j>, where j=1, 2, 3, 4. The value of k can range from 0 to 7.

a. If k = 0 the port is turned off. This is done by dropping DTR, which forces the modem to hang up if it is off hook and to not answer any further calls. The actual command to the modem is sent by the main program using the value returned from here.

b. If k = 1, the port is turned on. This is done by raising DTR, which causes the modem to execute an internal reset. The modem will now answer calls again.

c. If k = 2, the modem is set to NOHANGUP. A call will never be disconnected by the software and will continue transmitting time messages until the client hangs up either by sending the "%" character, by physically hanging up the line or by sending the "?" character. In this latter mode, the software sends the help message and hangs up when it has been completely sent. This state is cleared using command 0 followed by command 1 to reinitialize the modem and the line driver.

The return value is 64 + 3 bits for the command + 3 bits for the port number – 1. If the command bits are CCC and the port bits are PPP, the response is 7 bits:

1 CCC PPP

/DISPLAY, abbreviated as /d

This command refreshes the display. It does this by calling subroutine *setscn*.

/HALT, abbreviated as /h

This command stops the software and returns control to DOS. Any calls in progress are terminated and all ports are disabled so that they will not answer any more calls. The halt is implemented by returning 128 to the main program which actually closes the ports.

/LEAP, <j > abbreviated as /l, <j >

This command sets the leap second flag to j, where j can be

0 to disable the leap second flag,
1 to add one second on the last day of the current month
2 to delete one second on the last day of the current month

This command sets the flag in the output message and also enables the internal leap second code so that the specified leap second operation will be performed on the local clock. The change in the time code takes effect immediately. The leap second flag is reset to 0 if the system reboots or after the leap second occurs.

/UT1, <j > abbreviated as /u, <j >

This command sets the UT1 value in the time message to j in units of 0.1 s. The value of j can range from –9 to +9 corresponding to UT1 values of –0.9 s to +0.9 s, respectively. The change in the time code takes effect immediately. The UT1 value entered in this way is lost if the system reboots; the system uses +0.1 as a default value.

Name: dstflag

Usage: int dstflag(iyr,imo,idy)
int iyr;
int imo;
int idy;

This subroutine computes the daylight saving time flag that should be transmitted on the date specified by the calling parameters. The return value is used directly in the transmitted time message. The flag is an integer and uses the ACTS definitions:

= 0 means standard time (ST) is in effect
= 50 means daylight saving time (DST) is in effect
= 51 means transition from ST to DST is at 0200 today
= 1 means transition from DST to ST is at 0200 today

< = 49 and > 01 means now on DST, go to ST at
0200 local time when the day is 01.
< = 99 and > 51 means now on ST, go to DST at
0200 local time when the day is 51.

The subroutine assumes that the transitions occur on the first Sunday in April and the last Sunday in October. It provides advance notice from 1 March for the April transition and from 1 October for the October transition. This is less notice than the original ACTS system provided, but the computation can be simplified using this system since there is no need to compute the advance notice value for February (which can have either 28 or 29 days due to leap years and is therefore a complicated business.) This method provides at least 30 days of advance notice in the Spring and 24 days in the Fall and that is probably enough anyway since most clocks cannot really use the advance notice until the last day of the month anyway.

The subroutine first examines the month value and uses the following method to compute the flag:

1. November, December, January, and February are always on standard time and no advance notice will ever be needed during these months.

2. May, June, July, August, and September are always on daylight saving time and no advance notice will ever be needed during these months.

If the month is March, April, or October, these rules do not specify the answer. The subroutine then examines the month and the day together:

3. All of March and the first days in April will always require a Spring count down; most of October will require a Fall count down. In both of these cases, the countdown algorithm is applied for the whole period with a test to see whether it has hit the transition date.

The count down algorithms uses the day of the week for 1 March as the reference point for computing the countdown values. There are seven possible values which are true for both leap and regular years, and the year number can be computed using ***cvt2jd()*** to find the MJD of 1 March followed by ***idaywk()*** to find the day of the week corresponding to that MJD:

Year #	Day of 1 March	First Sunday in April	Last Sun in October
1	Sun	5 April	25 October
2	Mon	4 April	31 October
3	Tue	3 April	30 October
4	Wed	2 April	29 October
5	Thu	1 April	28 October
6	Fri	7 April	27 October
7	Sat	6 April	26 October

Then compute the count down value for each one of these possibilities using the type of year from the previous table and the day of the month:

Year #	March Countdown	April Countdown	October Countdown
1	37-day	6-day	26-day
2	36-day	5-day	32-day
3	35-day	4-day	31-day
4	34-day	3-day	30-day
5	33-day	2-day	29-day
6	39-day	8-day	28-day
7	38-day	7-day	27-day

After applying the count down formula for April and October, check to see whether the formula results in a value less than the transition number (51 in April and 1 in October). If it does then the transition has happened and the flag is clamped at the steady-state value of 50 and 0 for the remainder of April and October, respectively.

This method can be modified for other transition dates using 1 March as the reference date and recomputing all of the values in the table for the transition months.

Name: getmsg

Usage: void getmsg()

This subroutine gets the help and welcome messages from two files named MESSAGE.HLP and MESSAGE.WEL. The welcome message is sent to each user as soon as the modem reports that the connection is complete, and the help message is sent if the caller types "?" at any point when the time messages are being sent. It also reads

the column headings from two files named MESSAGE.HIS and MESSAGE.LOS for the headings sent at high and low speeds.

The messages are copied into global arrays *welcome*, *helpmsg*, *hdrhisp*, and *hdrlosp*, respectively, by this subroutine. The subroutine is called only when the software starts; memory resident copies of the messages are used after that.

The welcome and help messages can be up to 300 characters long and 3000 characters long, respectively; the two column headings can be up to 90 characters each. Longer messages can be used if the lengths of the arrays are adjusted and the software is recompiled. The messages can consist of any of the usual printing characters whose ASCII values are between 32 decimal and 127 decimal. In addition, ASCII 7 (bell) will be passed, ASCII newline (10 decimal) will be transmitted as carriage return, line-feed (10 decimal followed by 13 decimal), and ASCII tab (9 decimal) will be expanded to enough consecutive spaces to advance the cursor to the next "tab stop." Tab stops are located every eighth space starting with col 1. The tabs stops are in columns: 1, 9, . . . All other control characters in the input files will be ignored.

Name: idaywk

Usage: int idaywk(day)
long int day;

This subroutine returns the day of the week corresponding to the MJD specified in input parameter day. The return value is an integer from 1 to 7, where 1 means Sunday, and so on.

Name: inivrf

Usage: int inivrf()

This subroutine initializes the network sockets to link to the verify time service via the UDP port specified in parameter *pserv* located in the file MACHINE.H. This client process sends a message to the server. The four character response shows the difference between client time and server time, (client – server), in seconds.

This version uses the WATTCP public domain library of functions to interface to an existing packet driver; the packet driver is also part of the WATTCP library. In addition to initializing the socket code, the subroutine converts the address of each server from the usual "dot" form to the internal binary form. The return value is always +1.

Name: kybcmd

Usage: int kybcmd()

This subroutine checks the keyboard buffer for a character. If no character is present it returns 0. If characters are present, it stores them in global array *kbuf*. If the subroutine reads a newline character it adds a trailing 0 byte and returns a status of 1 to the caller, showing that the message is complete. If it does not read a newline character, then the message is not yet completed. It stores the characters in the buffer, advances the buffer pointer and returns 0. If a backspace character is read, the subroutine decrements the buffer pointer by 1, thereby "erasing" the previous character. Consecutive backspaces will continue to decrement the pointer until the entire line is erased.

The subroutine returns a value of –1 if it reads a newline character as the first character of the line. This can happen if a carriage return is entered by itself in response to the command prompt or if it is entered after a string of characters which have been completely erased using the backspace key. There is nothing to parse in this case, so that the buffer points are reset and the command prompt is repeated.

Name: putcur

Usage: void putcur(irow,icol)
int irow;
int icol;

This subroutine positions the screen cursor to position *irow* and *icol* specified in the input parameters. The operation is performed using interrupt 10h.

Name: secnds

Usage: long int secnds()

This subroutine returns the current system time as the number of seconds after midnight. It uses interrupt 1Ah to read the time in ticks and converts this value to seconds using the standard conversion factor of 18.2064893 ticks per second. The value is truncated and converted to a long integer and returned to the caller.

Name: setscn and setscnl

Usage: void setscn()
void setscnl()

This subroutine sets up the screen display. It is called when the software starts and whenever the screen needs to be repainted as specified by the /d command from the keyboard. The subroutine contains constants that define all fixed fields that are part of the display. The –l variant of the subroutine is used for the LOS version of the server (Appendix A). It is identical except that the title line identifies the software as ACTSLOS.

Name: timcod

Usage: void timcod()

This subroutine generates the time code. The subroutine generates two time codes, a high speed code that is stored in array *obuf*, and a low speed code that is stored in array *obuf3*. Both arrays are defined as globals in the main program. The time codes are composed of a universal portion consisting of the MJD, date and time, followed by a number of optional fields such as the DST flag, the leap second flag, and the UT1 value. These fields are enabled in the message using constants defined in the header file named MACHINE.H.

The low speed time code does not have the MJD or UT1 fields but is otherwise the same as the high speed code. The seconds value of the low speed code is set blindly to one second more than the value in the high speed code, even if doing so results in an illegal time. As discussed in the section on principles of operation, only the codes

that have odd seconds are actually transmitted. Although we could save time by not generating codes that will not be used, the software would not necessarily be faster or smaller since the test for whether the second was odd or even would require some "if" loops and possibly an additional flag to show that the second was even.

The time required to generate the full code is about 100 μs, which is small compared to the 1 s interval between messages. Therefore there is no point in exploiting the fact that much of the time code is the same from second to second and that we could save some time by only changing the new bit of the code.

The time code is derived from the system time. The date and time are obtained directly from the system and the various other bits of the code are computed using ***cvt2jd()*** to compute the MJD from the date and ***dstflag()*** to compute the daylight saving time flag from the MJD. The leap-second flag and the UT1 correction are specified by external commands.

If the "leap second today" flag is set then this subroutine will insert the leap second into the code on the last second of the day. This flag is set by ***advdat()***.

⇒ Time sequence that occurs when inserting the leap second:

23:59:59 UTC
23:59:60 UTC
00:00:00 UTC

In addition to generating the time code, the subroutine puts the binary value of the hour in the last location of the *obuf* array (which is *obuf*[79], since the array is 80 bytes long). This is used as the index for the call counters; when a call is received, the value in the location is combined with the port number and the pair is used to specify which call counter will be incremented.

Name: tos

Usage: int tos()

This subroutine returns the tick of the current second. The system clock counts in ticks, where there are 18.206 ticks/s. The returned value ranges from 0 to 18.

The subroutine reads the system time using interrupt 0x1Ah and computes the tick as the system time modulo 18.2064893.

Name: unpbcd

Usage: int unpbcd(ival)
int ival;

This routine receives a packed BCD value in ival (as read from the CMOS clock) and returns the integer value. It is the inverse function to ***cnvbcd()***, which produces a packed BCD value from an integer input.

Name: wrtmsg

Usage: void wrtmsg(irow,icol,bufi,mode)
int irow;
int icol;
char *bufi;
int mode;

This subroutine writes the ASCII string stored in bufi to the screen starting at position *irow*, *icol*. The subroutine writes the message directly to the screen memory addresses starting at the values corresponding to positions *irow* and *icol* and continuing consecutively. The message is terminated by either a 0 byte or a <return> character. If a 0 byte is used, the routine writes the message and simply returns, but if a <return> is found the remainder of the current line is filled with spaces. Thus a shorter message can be written over a longer message.

The mode parameter specifies how the message will be displayed. On a monochrome screen:

mode = 2	normal text
mode += 8	Bold text
mode += 1	Add underline
mode += 128	Blinking
mode += 112	Reverse Video

The VGA color version maps the modes above using colors as follows:

input mode =2, normal text => 15 High intensity white
input mode =10, bold text => 14 High intensity yellow
input mode =1,3 normal underline => 10 High intensity green
input mode =240 reverse blinking => 200 Reverse red blinking

Other input modes are not changed.

This mapping is done in this routine so that the rest of the code does not have to be changed if the monitor is changed.

Machine Configuration File

Most of the parameters needed to configure the software are located in file MACHINE.H. The normal values stored in this file are shown below together with an explanation of how to change them if needed:

#define VGA 1	Set this parameter to 1 if the machine has a VGA color display, or to 0 if it has a monochrome display.
#define DST 1	Set this flag to 1 if machine transmits DST flag as part of the time code message, or to 0 if it should not.
#define LS 1	Set this flag to 1 if machine transmits LS flag as part of the time code message, or to 0 if it should not.
#define UT1 1	Set this flag to 1 if machine transmits UT1 flag as part of the time code message, or to 0 if it should not.
#define ADV 1	Set this flag to 1 if machine transmits advance time as part of the time code message, or to 0 if it should not.
#define NAME 1	Set this flag to 1 if machine transmits the name of the UTC time code as part of the message, or to 0 if it should not.
#define NCOM 4	Set this parameter to the number of COM ports. The maximum value is six.

For each COM port specified above, the following parameters give its hex address and IRQ values (the values below are for the four modem server):

```
#define ADR1 0x280
#define ADR2 0x288
#define ADR3 0x290
#define ADR4 0x298
#define IRQ1 4
#define IRQ2 3
#define IRQ3 10
#define IRQ4 11
```

The following parameters specify the offsets from the COM port base address to the modem control register (MCR) and modem status register (MSR). Normally these values should not be changed.

```
#define MCR 4
#define MSR  6
```

The following arrays are used as I/O buffers. There must be one for each COM port. The bytes are divided as 132 for output and 132 for input. Although most of the transmissions are output only, the input buffer is made large enough to deal with the modem response strings, which may come very close together.

Arrays *comadr* and *comirq* hold the definitions of the COM-port addresses and interrupts defined above.

These arrays are defined only in the main program. This limitation is handled by the if-define of MAINEP, which is only in the main program.

```
#ifdef MAINEP
char combuf[NCOM][264];
static int comadr[NCOM] = {ADR1,ADR2,ADR3,ADR4};
static int comirq[NCOM] = {IRQ1,IRQ2,IRQ3,IRQ4};
#endif
```

If VERIFY is defined, then this version checks its time using UDP/IP transmissions to the verify service of one of the NIST time servers. The number of verify servers is defined by the value of VERIFY. For each VERIFY server, parameter srvdot gives the internet address(es) of the machines in standard dot (.) notation; these values are converted into binary notation by the program and the results are stored in the corresponding locations of array *srvad*.

These arrays are selectively defined via the symbol MAINER, which is defined only in the network-based subroutines, and MAINEP, which is defined in the main program as well.

```
#define VERIFY 2
#ifdef VERIFY
#ifdef MAINER
static char *srvdot[VERIFY]={"132.163.135.130","192.43.244.18"};
long int srvad[VERIFY];
int pserv = 1123;      UDP/IP port for verify service. Should not be changed.
#endif                 end if for MAINER
#ifdef MAINEP
int vrfdue;            flag to show verify in progress, maintained by program.
int netdif[VERIFY];    time difference to k-th server, maintained by program.
#endif
#define VRFINT 300 interval in seconds between verify cycles.
#endif /*VERIFY*/
```

If the parameter GT is defined, then the hardware configuration includes a Guide Technology GT401 Event timing and controller board.

```
#define GT 1
#ifdef GT
#define GT_ADDR 0x100          Hardware address of board

#define GT_INT 100             Interval between comparisons of the DOS clock
                               and the board time, in seconds.
#ifdef MAINEP
int gtdue;                     Countdown to next comparison, maintained by
                               the program.
#endif
```

#define HITOL 0.82	Upper limit of difference (DOS clock – Guide board), in seconds. This value is 15 DOS clock ticks (rounded up).
#define LOTOL 0.71	Lower limit of difference (DOS clock – Guide board), in seconds. This value is 13 DOS clock ticks (rounded down).
#endif	
#define MSDELAY 33	The software will estimate the telephone line delay if this parameter is defined by looking for the echo of the OTM from each time code transmission. The value of MSDELAY must be the position in the time code message where the advance is transmitted, since the advance value is changed by the main program for each transmission from each port. The normal value is 33, but it may be different if some of the optional portions of the time code are not used.
#define NOMDELAY 45	The nominal advance of the OTM to be used if the user does not echo the OTM or if the echo times are not stable. The normal value is 45 ms. This value is about right for most 1200 bit per second modems, but may be too small for modems that use higher speed protocols.
#define MAXDELAY 205	The parameter MAXDELAY specifies the approximate value for the maximum advance that can be supported by the program. This value is used for the delay if the apparent OTM round-trip time appears to be 0 or small, which is interpreted to mean that the delay is so large that what is actually being observed is the OTM echo from the previous second. See the comments in the main program ACTS.C for more details.

#define SP300 23	If parameter SP300 is defined, then the program will support users who connect at 300 bits per second. The time code is shorter in this case and is sent only every other second. If measured delay mode is used then the value of this parameter is defined to be the position of the advance parameter in the output time code just as with MSDELAY above.
#define OLDMODEM 1	If parameter OLDMODEM is defined then the server is connected to the telephone system via dual-speed 300/1200 modems. The effects of this switch are described in Appendix A.

Parameter MINADV specifies the minimum advance that will be accepted as a valid one-way delay measure. This is typically 30 ms at speeds of 1200 bits per second and lower and 20 ms at higher speeds. The advances are specified in absolute milliseconds; a value of 970 thus implies an advance of 30 ms.

```
#define MINADV 970      (MSDELAY and OLDMODEM defined)
#define MINADV 980      (only MSDELAY defined)
```

In addition, if the network verify option is used, the internet address of the ACTS server must be specified in a file named WATTCP.CFG, which must be located in the same directory as the software itself. The format of this file is shown below. Each line begins with a key word that must be entered exactly as shown followed by a value.

my_ip= Insert ip address of machine here in "dot" notation.
Example:

my_ip=140.141.142.143

netmask= Insert netmask for local network in "dot" notation.
Example:

netmask= 255.255.255.0

gateway = Insert ip address of gateway in "dot" notation, if necessary. Example:

gateway = 140.141.1.1

nameserver = Insert ip address in "dot" notation of machine that can provide name resolution using Domain Name Service. This keyword can be repeated for multiple servers. Example:

nameserver = 140.141.1.2
nameserver = 140.141.1.3

domainslist = Insert name of domain for this machine. Example:

domainslist = "nist.gov"

Chapter 5

Technical Reference for ACTS Monitor

The ACTS Monitor (ACTSM) is an accessory to the ACTS system. It performs the following functions:

- Compares the ACTS time codes to the time codes generated by its own independent clock (REF_CLK)
- Inspects the availability of access to the ACTS servers
- Collects the statistics (number of calls) from the ACTS system

Unusual incidents are reported in log files. If ACTSM detects a date/time error (the time difference between ACTS and ACTSM exceeds the specified limit), an alarm signal (open circuit) will be sent to the ACTS alarm system. If no ACTS server can be reached during a scheduled inspection, an alarm message is displayed on the ACTSM screen. The ACTSM can monitor up to four ACTS servers. The ACTSM will handle the leap year automatically. Leap seconds will be taken care of according to the leap second flag status in the ACTS time codes.

Hardware and Software

The ACTSM hardware consists of a PC (with either a 486 or Pentium processor), an ACTSM ISA bus card made at NIST, the monitor eavesdrop box, a 1200 bit per second (or faster) modem, and an optional Ethernet card for connection to the Internet. The ACTSM board contains the REF_CLK, I/O port interface circuit and the circuit required to synchronize the REF_CLK to the external reference 1 pps. The REF_CLK requires a very stable 5 MHz or 10 MHz reference signal, preferably from an atomic oscillator. The external reference 1 pps is required only while setting the REF_CLK.

The software of ACTSM consist of three executable programs and a parameter file called ACTSMON.DAT. SETCLK2K.EXE is used to synchronize the REF_CLK to the external 1 pps. SETUPMON.EXE is used to configure the ACTSM.

ACTSM2K6.EXE (or a similar name) is the monitor operating program. The software is written in BASIC. Although the software is compiled with QuickBASIC (Version 4.5), the software can also be run in QBasic under MS DOS (Version 5.0 or later). The software is written for DOS and might not work properly using Microsoft Windows or another multi-tasking environment.

Configuration Parameters

The ACTSM configuration file is named ACTSMON.DAT. The configuration file includes:

- The modem initialization command
- The communication parameters (including the COM port and baud rate)
- The telephone numbers of each server enabled for monitoring
- The number of ACTS time codes to be received when ACTSM calls a server
- The interval between inspections of the ACTS servers
- The maximum allowable date/time difference between ACTS and ACTSM before an error message is generated.

The configuration can be modified by SETUPMON.EXE, and viewed while running the monitor software. Here are some suggested settings:

- Inspect all servers every 10 minutes.
- Receive 10 time codes during each inspection of a server.
- Define a date/time error as | REF_CLK - ACTS | >= 100 ms. Keep in mind that a date error is clearly more than 100 ms.

Operation of the ACTSM

Because REF_CLK requires DC power from the PC bus and an external reference frequency to operate, it needs to be set on time whenever the DC power or the signal from the reference frequency source has been interrupted. In other words, if you power down the monitor computer, or disconnect (even for an instant) the cable from the external reference frequency, REF-CLK needs to be reset. REF_CLK should be set within ±0.5 s of UTC. To do so, run SETCLK2K.EXE. Select 1 to dial ACTS and synchronize the clock to the external 1 pps, or 2 to synchronize the clock to the external 1 pps without calling ACTS. When finished, press <ESC> to exit.

To start the monitor software, run ACTSM2K6 (or a similar name) from the DOS prompt. This section describes how the monitor software works.

The ACTSM monitors the server in two different ways, by eavesdropping on the server's serial lines or by calling the server through a modem and telephone line (in the same fashion as any ACTS customer). The monitoring operation can be divided into the Idle state and the Monitoring state. In the Idle state, the ACTSM performs these tasks:

- Reads and displays the REF_CLK and DOS clock (the DOS clock is updated every two hours to agree with the REF_CLK).
- Responds to commands entered by the user (will be discussed later).
- Waits for the next inspection of ACTS (entering Monitoring ACTS state).
- Downloads statistics from ACTS servers at the beginning of each UTC day. If the monitor is connected to the Internet, it can then upload (via FTP) these statistics files to a web server so that they can be viewed remotely.

During the inspection of a server (Monitoring State), the ACTSM performs these tasks:

- Schedules the next round of inspections.
- Devotes half of the time interval between two inspections to eavesdrop on the serial lines of all enabled servers. After the eavesdropping period is over, ACTSM will call the enabled servers which were not reached during the eavesdropping period. Although an ACTS server can have up to six lines, the time codes sent to each line are generated by the same server. Therefore, ACTSM eavesdrops only on the first line of a server. Eavesdropping only works if a client is connected to the line during the eavesdrop period.
- Displays and records the received time codes.
- Computes the date/time difference between the ACTS server and REF_CLK.
- Displays the server ID, the number of # sign OTMs received (when ACTSM dialed ACTS) or the number of time codes received (through eavesdropping), the date and time portion of the last received time code, and the time difference between the server and REF_CLK

- Verifies that the date and time portion of two consecutive time codes differs by only 1 s (a code error is declared if the difference is not 1 s)

- If three consecutive date/time error conditions occur, an alarm signal (open circuit) is sent to the ACTS alarm system

Monitor ACTS	Selected by pressing "**M**" or "**m**". This starts monitoring ACTS immediately even if an inspection is already scheduled. When the ACTSM software is first started, it stays in the Idle state until this function is selected, and then it automatically switches between the Idle and Monitoring states.
Inspect	Selected by pressing "**I**" or "**i**". This starts inspection of the server specified by the operator. ACTSM will respond by asking the operator to enter the server ID. ACTSM will spend 30 s attempting to eavesdrop on the serial line. If there is no activity during that period, ACTSM calls the server via the modem and telephone line.
View setup	Selected by pressing "**V**" or "**v**". This displays the current configuration of ACTSM.
DOS Shell	Selected by pressing "**D**" or "**d**". This pauses the monitoring software, and passes control to DOS. The primary use of this function is to access the error trapping log files. Typing "EXIT" at the DOS prompt restarts the monitor software. If it has passed the scheduled inspection time when returning back from DOS, the monitor starts inspecting ACTS immediately.
Reset	Selected by pressing "**R**" or "**r**". This resets the alarm switch and the alarm message displayed on the ACTSM screen.
Exit	Selected by pressing "**E**" or "**e**". This exits the ACTSM software and returns to DOS.

There is also an unlisted function called by pressing "**T**" or "**t**". This function downloads the statistics to the ACTSM screen for maintenance purposes.

During the eavesdropping period of the Monitoring state, ACTSM will check for a keyboard interrupt (<ESC> keypress) every 30 s. When ACTSM calls the server, it will check for a keyboard interrupt every time it finishes a call.

If all enabled servers have been successfully checked, ACTSM switches to the Idle state. If ACTSM cannot inspect a server within the time interval specified in ACTSMON.DAT, the incident is recorded in the log file. If no server has been reached during the inspection (for example, if the telephone lines are out of service or the ACTS servers are down), an alarm message is displayed on the ACTSM screen. The alarm signal can be reset by selecting the Reset function from the keyboard (pressing R) or by restarting the software.

In the Idle state, the ACTSM responds to input from the keyboard. In the Monitoring state, you must press the <ESC> key before entering data from the keyboard. There are six selectable function calls as listed in the table on the previous page.

Screen Display of the ACTSM

The ACTSM screen is divided into five areas as shown in Figure 3.

The top of the screen is the header area. The middle of the screen displays time codes from as many as four ACTS servers. Each ACTS server is assigned to a time code window. Each time code window consists of four rows. The first row displays the server number and the number of "# sign" OTMs received (when ACTSM calls ACTS) or the number of time codes received (when ACTSM is eavesdropping). The second row displays the date and time portion of the last received time code. The third row displays the REF_CLK stamp of the last received time code. And finally, the last row displays the time difference in milliseconds between the server and REF_CLK. In the case of a date/time error, the time difference is displayed in red and blinks.

The display in each time code window is updated when an inspection to the server is completed. The Monitor Operation Status window in the right lower corner is used to display diagnostic messages. Descriptions of the Monitor Operation Status messages are given in Appendix D. Under the time code area are the REF_CLK and DOS clock displays. In the Idle state, these displays are updated approximately every millisecond. In the Monitoring state, they are updated approximately every 30 s.

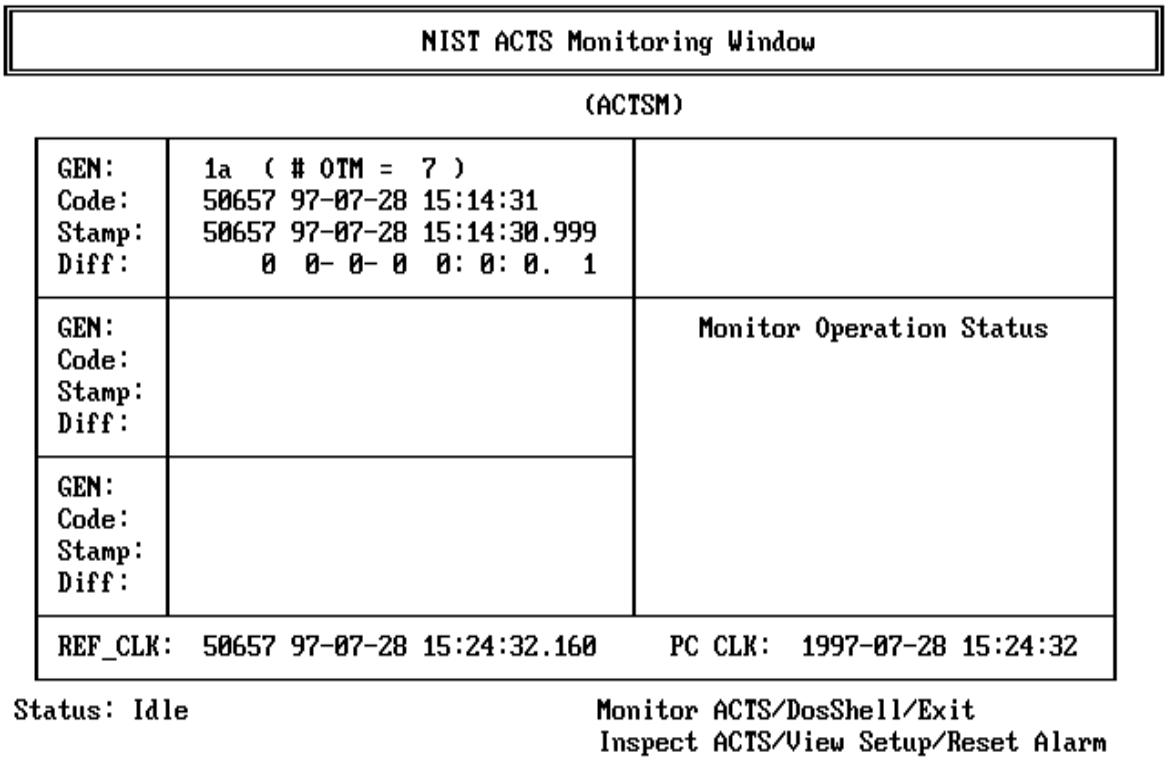

Figure 3. The ACTS Monitor Screen Display

The lower right corner contains the menu of function calls. The ACTSM status is displayed on the lower left corner of the screen. In the Idle state, a status display example is given as shown in the picture above. The "next call" information will not be displayed if the Monitoring state has never been entered. In the Monitoring state, the status display reports the progress of inspection, and the time codes received.

The status display reports the progress in the following sequence, where x is the ACTS server to be called; ii is the number of calls (from 1 to 10) made in the current inspection; jj is the number of date/time errors (from 0 to 3) detected; and y is the connected ACTS server (y might not be the same as x).

Status: Monitoring ACTS
Target: Bank x, (#ii-jj)
Status: Open Communication Buffer
Target: Bank x, (#ii-jj)

Status: Initialize MODEM
Target: Bank x, (#ii-jj)
Status: Calling ...
Target: Bank x, (#ii-jj)

Status: Bank y connected
Header received
Status: Bank y connected
1 time code(s) received
(a complete ACTS time code with OTM)
:
:
Status: Bank y connected
10 time code(s) received
(a complete ACTS time code with OTM)

Error Handling and Log Files

The ACTS monitor records errors by writing to a log file when an error occurs. All possible errors are grouped into seven categories, and each has a corresponding log file. The log files all have the extension of ".ACT" or ".AC2", depending upon the configuration. The content of each log file is discussed below.

ERR_H.ACT— This file records all run-time errors, such as the DEVICE I/O ERROR, DEVICE TIME OUT, and so on. When a run-time error occurs, the error code and time stamp are recorded.

COMM_ERR.ACT—This file records when the communications buffer is empty for more than 30 s and when the MODEM reports an error message, such as BUSY, NO CARRIER, and so on. In either incident, the server number, the error message and the time stamp are recorded.

TIME_OUT.ACT—This file records the ACTS server(s) that could not be monitored during the time interval between inspections. Each list contains the time stamp at the end of the inspection, and the server number.

CODE_ERR.ACT—This file records groups of all the received time codes that contain an error.

TIME_ERR.ACT—This file records date/time errors. The messages recorded contain the server number, the time stamp when the date/time error is computed (last time code received) and the date/time difference.

ALARM.ACT—This file records:

1. when three consecutive date/time errors are made by the same ACTS server; and
2. when no ACTS server was reached during the time interval between two inspections.

In the case of date/time error alarms, the recorded message includes the server number, the time stamp when the third date/time error is computed, and the date/time difference. In the other case, the message records the time stamp of the last trial and the information that no bank has been connected in the time interval.

OTM_x.ACT—This file records when fewer than three # sign OTMs were recorded (x denotes the server number).

Besides the error log files, the monitor logs the total number of phone calls made to each server in files named "yymmSTx.ACT", where yy is the year, mm is the month, and xx is the server number.

Handling Leap Seconds

The monitor checks the leap second flag in the ACTS time code each time it inspects a server. If nonzero-valued leap second flags from all the servers agree with each other, ACTSM will set its own leap second flag and compute the MJD when the leap second is scheduled. ACTSM will display the leap second message on its screen, and record the leap second flag rollover (from zero-valued flag to nonzero-valued flag or from nonzero-valued flag to zero-valued flag) in the log file named LEAP_SEC.ACT.

Five minutes before the leap second, ACTSM will pause its inspection of ACTS and wait for the leap second moment. After the leap second moment, ACTSM will reset its REF_CLK according to the non-zero valued leap second flag, and reset its leap second MJD, then resume the inspection of ACTS. Because ACTS servers should have reset their leap second flags by then, ACTSM will detect the change in the ACTS time codes and record the leap second flag rollover in the file named LEAP_SEC.ACT.

The ACTSM REF_CLK

The REF_CLK is an accurate, real time clock that serves as a reference time source to the ACTSM. The REF_CLK is a 12.288 MHz VCXO that is locked to an external 5 MHz or 10 MHz reference frequency. The 12.288 MHz signal is then divided by 375 to generate the 32.768 kHz time base for driving the real time clock chip (HM58167 or HM68167). The clock chip outputs the month, day of month, day of week (not used), hours, minutes, seconds, tenths and hundredths of seconds, and milliseconds. Each value can be accessed (read/write) at addresses 240h through 247h. The year and MJD are managed by the monitor software. The 5 MHz or 10 MHz external reference frequency can be selected by setting a jumper on the board.

The REF_CLK card also contains a relay for the ACTSM alarm. Under normal conditions, there is a 39 W resistor in series with the closed relay. The relay will switch to open circuit after receiving the alarm SET signal. The two relay output terminals are connected in series with the ACTS alarm system.

Uploading Files to an FTP Server

The ACTS monitor software has the ability to upload files to a remote server using the File Transfer Protocol (FTP). This allows the files to be archived on a remote server. A web server can even display the number of phone calls received by ACTS and the diagnostic files if a CGI (common gateway interface) script is used to automatically process the uploaded files.

In order to use this feature, the monitor must be connected to the Internet using a DOS compatible Ethernet card, and a DOS TCP/IP stack and FTP program must be installed. This feature has been successfully implemented using the Sockets TCP/IP stack for DOS available from Datalight (www.datalight.com). The information below shows how to configure the ACTS monitor to use Sockets. This information would need to be modified if another DOS based TCP/IP stack is used. For the purposes of this example, the Sockets software has been installed in the c:\socket directory.

The AUTOEXEC.BAT file must be modified to include the following two lines:

```
path c:/socket
socketp socket.cfg
```

The SOCKET.CFG file referred to in AUTOEXEC.BAT includes the following information:

```
ip address 132.163.136.91                ‘ IP address of ACTS monitor
interface pdr if0 dix 1500 10 0x62       ‘ packet driver interface using Ethernet
route add default if0 132.163.136.254    ‘ IP address of route to network
domain server 132.163.129.1              ‘ IP address of domain server
ip ttl 15                                ‘ maximum number of gateway hops
tcp mss 1460                             ‘ maximum TCP segment size in bytes
tcp window 2920                          ‘ maximum receive window size in bytes
```

The CONFIG.SYS file must be modified to include the following two lines:

```
device=de22xpd.sys 98
set sockets=c:\socket
```

The file de22xpd.sys is a driver included with the Ethernet card and and must reside in the root directory.

The monitor software calls the FTP program by issuing a command from the DOS prompt as follows:

```
ftp /v 132.163.4.82 /f=script.txt
```

where 132.163.4.82 is the ip address of the remote server, and script.txt is a file containing the dialogue with remote server. The script.txt file must contain the following information:

```
USER xxxx          ‘ where xxxx is the user name
PASS xxxx          ‘ where xxxx is the password
CD\xxxx            ‘ where xxxx is the directory on the remote server
PUT xxxx           ‘ where xxxx is the file name to be sent to the server
QUIT               ‘ end FTP session
```

Appendix A

Low-Speed ACTS System

The low-speed software is designed for ACTS servers that use the older dual-speed 300/1200 baud modems. These modems have delays that are smaller and more stable than the newer multi-speed modems. The dual-speed 300/1200 modems differ from the multi-speed modems used in two important respects:

- The interface between the computer and the modem must run at the same speed as the telephone line's transmission rate. The standard system uses a serial-line connection between the computer and modem that operates at a fixed speed of 19.2 kilobits per second.

- The modems do not have EPROMs, and any changes from the factory-default configuration must be reprogrammed each time the modem is reset. The standard system uses a reset command to reload a user-specified configuration.

The software differences that are needed to deal with these hardware differences are enabled by defining parameter OLDMODEM in the file MACHINE.H. The value of this parameter is not significant in this version, but this may not be true in the future. If OLDMODEM is defined, the minimum acceptable advance is increased from 20 ms to 30 ms by changing the parameter MINADV in MACHINE.H. The program ACTSLOS differs from the conventional ACTS server software as follows:

A. During startup initialization:

Set initial set of each line to state 106 (instead of state 0). This will reprogram the modem configuration on the next second.

B. Change to State 38, Action-2: (complete CONNECT message has been received)

If a low-speed connection is detected (modem response is CONNECT <1f>), then change the speed of the corresponding serial line to 300 bits per second. This action is in addition to the other actions described for this state.

If the connection is at 1200 bits per second (modem response is CONNECT 1200) then no speed adjustment is needed since the speed was already set during the previous hangup/reset sequence (see state 106 below).

C. Change to State 83 and prepare for next call on this line.

Action:

2. Advance to state 106 (instead of state 0) and end action (reprogram modem configuration on next second).

D. New States 106 and 107:

State 106: Reprogram modem configuration

Action:

1. Set speed of serial line to 1200 bits per second unconditionally; previous speed setting is lost.

2. Send modem configuration string as defined by parameters **resstg** (character pointer containing the string) and **lenres** (length of configuration string in bytes).

3. Advance to state 107 and end of action (delay for 1 s while configuration string is sent and requested parameters are stored in the modem).

State 107: Delay after reprogramming.

Action:

1. Flush input buffer (erasing OK response to configuration string sent in state 106).

2. Return to state 0 and end of action (return to idle state, wait for next call).

Note that actions 1 and 2 of state 106 insure that both the serial line and the modem are reset to 1200 bits per second following every call and at any time that a reset is performed via a call to states 82/83.

The low-speed software has been built and tested using Maxum external modems

model MX1200E, but similar modems from other manufacturers should work as well. The switch settings for the Maxum modems are shown in the following table.

Switch Number	State	Action
1	OFF	Support Data Terminal Ready
2	OFF	Full-Word Result Codes
3	ON	Enable Result Codes
4	OFF	Command Echo On
5	OFF	Auto Answer Enabled
6	OFF	Support Data Carrier Detect
7	--	Not Used
8	ON	Enable Commands
9	OFF	Use Bell 103/212A Standards
10	OFF	USA Dialing Intervals

The reset string sent in state 106 is:

at m0 x1 s0=1 <return> (spaces are inserted for clarity; the actual message length is 11 characters)

which configures the modem to:

m0: Turn loudspeaker off always
x1: Enable CONNECT 1200 response
s0=1: Answer calls on the first ring

The remaining parameters are set to their default values by the hardware reset performed in states 82 and 83.

Appendix B

Summary of Server Commands

This section summarizes all ACTS server commands. Additional details are in the section describing the operation of subroutine docmd().

/COM<j>,<k>, abbreviated as /C<j>,<k>

This command executes command <k> on COM port <j>, where j = 1, 2, 3, 4, 5, 6.

Commands:

k=0 Turn the port off. A call in progress is disconnected and the modem will not answer the telephone.

k=1 Turn the port back on. Calls will now be answered again.

k=2 Set the modem to NOHANGUP. A call will never be disconnected by the program and will continue transmitting time messages until the client hangs up. The status line changes to reverse video in this case. The NOHANGUP state is cleared using command 0.

/DISPLAY, abbreviated as /d

This command erases and redraws the entire screen. It is called automatically when the program starts and may be used after that if a glitch corrupts the screen display.

/HALT, abbreviated as /h

This command stops the software and returns control to the operating system. Any calls in progress are terminated and all COM ports are disabled.

/LEAP,<j> abbreviated as /l,<j>

This command immediately sets the leap-second flag to j, where j can be

0 to disable the leap second flag,
1 to add a second on the last day of the current month
2 to delete a second on the last day of the current month

/UT1,<j> abbreviated as /u,<j>

This command sets the UT1 value in the time code to j in units of 0.1 s. The value of j can range from -9 to +9 corresponding to UT1 values of -0.9 s to +0.9 s, respectively. The change in the time code takes effect immediately and is lost if the system reboots. The default value is +0.1 s.

/VERIFY,<j>, abbreviated at /V,<j>

This command enables or disables communication with verify server number <j>. If <j> is positive, comparisons with the corresponding server are enabled and if <j> is negative they are disabled. If the link to the server is already in the requested state then nothing happens. The absolute magnitude of <j> can range from 1 to 9, but only servers whose parameters were specified in file MACHINE.H at compile time can be configured. Thus, /v,3 will have no effect if only two servers are specified in file MACHINE.H. Likewise, the address and the other parameters of the server cannot be changed from the command line.

~s, command to check the server call counters

The "s" character must be received not later than 2 s after the "~". A "~" without a following "s" within 2 s will cause the program to hang-up and an "s" without a leading "~" will be ignored.

This command can only be sent through a COM port, it will not be recognized from the keyboard. It causes the values of all of the call counters to be transmitted. The counters are not reset by this command, and the call that submitted the request is counted.

The transmission consists of eight lines of text: each line consists of 24 three-digit values separated by spaces and terminated by a carriage return/line feed (4 ´ 24 + 2 = 98 characters total on each line). The first value on each line gives the number of calls received between 00:00:00 and 00:59:59, the second gives the number received between 01:00:00 and 01:59:59, etc. The time of a call is assigned when the initial connection is established.

The first four lines (or six lines, on six line servers) gives the call counters for the current day for each of the COM ports in order, and the second four lines give the same information for yesterday. Counters corresponding to times that are still in the future are set to 0. These arrays are cleared whenever the program is restarted.

The call counters are memory resident and are never written to disk by the ACTS server. They record only the last 48 hours, and will be lost if the server is restarted. Therefore, the ACTS monitor is responsible for periodically reading the call counters and saving them to disk for future reference.

Appendix C

Summary of Monitor Commands

The following commands are accepted by the ACTS monitor. These commands are executed by pressing the "hot keys" located in the lower right corner of the screen (note that A and T are not shown on the screen display). The function of each hot key is described below:

M Monitor all enabled ACTS servers immediately. The software spends the first half of period specified in the file ACTSMON.DAT (see Chapter 5) attempting to eavesdrop on the server's serial lines. If no action occurs in the first half of this period, the monitor will call the server by telephone.

I Inspect a specified ACTS server. The program will spend 30 s to eavesdrop on the first line of the server. If no action is detected, the monitor will call the server by telephone.

V View the contents of the configuration file, ACTSMON.DAT, on the screen.

D Go to DOS Shell.

R Reset alarm.

A Test the Alarm System. The monitor will simulate an alarm signal and notify the system operators with a phone call that transmits a series of tones. The alarm is then reset.

T Check statistics (number of phone calls per bank and line) for today and yesterday.

E Exit the monitor program.

Appendix D

Description of Monitor Operation Status

The following messages are displayed by the ACTS monitor in the Monitor Operation Status window:

Line 1 - Title	Monitor Operation Status
Line 2 - Alarm Status Indicator or "No Bank Connected in last 10 minutes" Indicator	Alarm!! No Bank Connected in last 10 minutes!
Line 3 - MJD of Leap Second and Leap Second Flag Status	Leap MJD, Leap Flag
Line 4 - reftimer = (reftime+time interval for inspection) and Elimit = half of time interval between inspections	reftimer, Elimit
Line 5 - reftime = # of seconds elapsed in current UTC day when inspection begins	reftime
Line 6 - Timer count down for inspection in progress or Eavesdrop termination prompt or Call termination prompt	timer Press Esc to terminate eavesdrop Press Esc to terminate the call!
Line 7 - Buffer Status on Eavesdrop	Buffer empty for 1 s
Line 8 - Exit Status Indicator	Exit from .....
Line 9 - Handler Error Indicator shows run-time error number	Handler Error