UNITED STATES DEPARTMENT OF THE INTERIOR

GEOLOGICAL SURVEY

HVO Polling telemetry system for low frequency data acquisition:

Software users' guide

by
Thomas T. English [1]

Open File Report 87-633

[1] Hawaiian Volcano Observatory
P.O. Box 51
Hawaii National Park, HI 96718

1987

Contents

I. System Overview

This report presents an overview of the HVO polling telemetry system, a description of the software portion of the system, and a guide to using the software. Program source code is included in an appendix.

The telemetry system consists of several components, including standard off-the-shelf hardware and custom designed hardware. This hardware operates under the control of an Apple microcomputer running in the UCSD-P environment. The Apple handles polling and data collection chores, and uses floppy diskettes for temporary data storage. It also transmits the collected data through an RS232 port as it is collected, making it available for real-time analysis and storage by other computer systems.

Station List Controls Polling

A station list controls polling operations. The list contains station names, identification numbers, radio frequencies for polling, the number of channels of data expected from each station (1 to 8), the polling period (1 minute to 1 hour), and a subnetwork identification code. The station hardware serial number, the bit count for a zero voltage level, battery conversion factor, and battery status flag are also included in the station list. Program STALIST maintains the station list in the Apple.

Polling Program MULTIAC

The polling program, known as MULTIAC, operates without intervention, and executes automatically when power is applied to the system. Initial operations include loading the station list from diskette, preparing pointers for subsequent data storage, resetting of control words for the external radio transmitter and receiver, and optionally establishing a data communication link with another computer. After it performs these initialization tasks the program waits in a loop until the seconds portion of an internal clock reaches zero. Then the station list is scanned to determine which stations need to be polled at that time. Polling is done by turning on either one of two radio transmitters, sending out the station's three digit code, then turning the transmitter off.

MULTIAC expects a response from the polled station within 2.25 seconds. When the response arrives, it is decoded and checked for consistency. The response comes in as a serial bit stream which includes a predetermined pattern of start and stop bits. If any of these bits are in the wrong state an error is flagged. Other error conditions include an incorrect station identification in the received data, incorrect channel numbers and failure to respond within 2.25 seconds. If any of these errors is detected then the polling and receiving process is repeated, up to a maximum of five times. Information concerning errors and retries is kept in an error log on diskette. Finally, the data are stored on the diskette and sent out thru the RS232 port.

MULTIAC Screen Displays

Screen displays during the polling process vary. The top line of the screen usually displays the current system status and the amount of space used on the diskette. Here is a list of the status codes and corresponding interpretations:

Code	Meaning
S	System is in the startup process.
W	System is waiting between polling cycles.
P	System is polling a station.
R	The receive operation is active.
I	The program is interpreting or decoding received data.
D	The data are being written on the diskette.

During the polling and receiving process, the screen shows the data flowing through the system. For each channel, the station identification, channel number, and raw data value are shown. The date and time are shown after each station. The seconds portion of the time represents the number of times the station has been polled in the current cycle.

After the stations for a given polling period have been processed, a status report of all stations in the system is displayed. This report includes the station name and the number of polling tries required during the last cycle. A zero for the number of tries means that the station has not been polled since the last system startup.

Polling Schedule Changes

The polling program accepts changes to the polling schedule specified in the station list. These changes are implemented on a subnetwork basis and stay in effect either until they are overridden by another subnetwork change or until the polling program is terminated for any reason. Permanent schedule changes can be made by changing the station list. The program can also poll any station upon operator request.

Guide to Remainder of Document

The remainder of the users' guide follows the same general sequence established in this overview section. The following sections deal with maintenance of the station list which controls system operation, normal operation of the polling program, special keyboard commands, error handling and logging features, and, finally, utility functions. The user should thoroughly read each section in order to become familiar with the system, and to get acquainted with this manual.

Each item in the table of contents is headlined in the body of the manual. This provides the user with a quick means of finding information later on.

II. Station List Maintenance

II.A. Station List Description

The operation of the data acquisition program is controlled by a station list that must be prepared prior to attempting normal system operation. This station list contains information regarding station names, identification codes, polling periods, and miscellaneous control information for each station known to the system. The various fields in the station list are discussed below. Then the station list maintenance procedures are presented.

II.A.1. Station Name

The first item in the station list is for a four letter station name. This name must not be duplicated anywhere on the station list. It must be unique for each station that is known to the system and should consist of alphanumerics only with no imbedded spaces or special characters. This name is used to refer to the station throughout the system.

II.A.2. Station Identification Number

The next item is a station identification number. This identification number, falling in the range from 0 to 255, is used to poll the station. It is the same number that is encoded in the field station for both polling recognition and data response. There cannot be any duplication of station identification numbers within the same radio frequency and for the sake of clarity, there should not be any duplication of identifications within the system.

II.A.3. Polling Transmitter Frequency

The system is capable of dealing with two radio transmitters, and the next item specifies the radio frequency code to be used. The appropriate reply is either a 1 or a 2, corresponding to F1 and F2 transmitters respectively.

II.A.4. Polling Period

The next item allows for establishing the polling period for the station. The valid choices are 1, 2, 5, 10, 15, 20, 30 and 60 minutes. The selection here determines how often the station is to be interrogated.

II.A.5. Subnetwork Identification

There is a facility within the system whereby stations may be grouped together in subnetworks, and this the is purpose of the next item. This subnet identification can be used later as a basis of temporarily modifying the polling period for all stations in a given subnet. The identification number must be an integer greater than zero.

II.A.6. Number of Channels

Each station in the system is allowed a maximum of eight channels of data. The next item allows the number of channels to be specified. Make sure if the station is sending back battery status information for its telemetry equipment that there are an adequate number of data channels specified. The battery

status information occupies the next available channel after the normal data channels. This is discussed in more detail below, under the battery status flag field.

II.A.7. Multiplexor Serial Number

All of the fields discussed so far are essential to the task of gathering and recording user data. There are several more items of information that are available for use at the option of the technicians who maintain the hardware for the data gathering functions. The first of these items is the serial number of the analog multiplexor card in the field station. This must be a positive integer.

II.A.8. Zero Volts Bit Count and Battery Conversion Factor

The next three items deal with reporting and interpreting battery status information from the field telemetry stations. They consist of a zero volts bit count, a battery conversion factor and a battery status flag. The zero volts bit count is a positive integer which specifies the bit count that corresponds to a voltage level of zero (i.e., dead short to ground) for the station. The battery conversion factor, also a positive integer, is used to convert the reported battery voltage from bits to an actual voltage. It's units are hundredths of millivolts per bit. For example, if the conversion factor for a given station is 7.6 millivolts per bit, the value is entered as 760 in the station list.

II.A.9. Battery Status Flag

The final field on the list is the battery status flag. This flag is set to 1 to indicate that the station does not report battery status, or to 2 to indicate that the last channel of information contains the battery status for the station. The convention here is that if a station is reporting three channels of user data (e.g., two components of tilt and a temperature) and battery status information is desired, then the battery status occupies the fourth channel. Similarly, if there are seven channels of user data, the battery status is in the eighth channel.

II.B. Program STALIST

The station list is built and maintained with the program STALIST, which must be invoked from the main command level of the P-System. There are a couple of ways of getting to the main command level. Initially this is accomplished by placing the acquisition program diskette in drive 1 of the system and leaving drive 2 vacant. The system automatically loads the acquisition program and attempts to find a valid recording file on the second disk drive. When this attempt fails, and an option message appears. The quit option is selected, bringing the system to the main command level.

When the acquisition program is in normal operation, the main command level of the P-System can be reached be entering the quit command. This will be discussed in more detail later on. Right now all that is important is to get into the station list maintenance program.

Once the system is at the main command level the station list program is invoked using the X command. The system prompts for the name of a file to execute, and the proper response is STALIST. The screen clears and a menu appears:

A(dd a new station

C(hange a station

D(elete a station

I(nitialize a new list

L(oad a list from disk

P(rint the list on the screen

Q(uit

S(ave the list on disk

Your Choice?

II.B.1. Add a New Station

The program automatically provides a clean slate to work with when it is first invoked, so stations can be added to the list using the Add selection. This selection results in prompts for the various items to be included for each station. The prompts are self-explanatory and the responses must be in accord with the station list description, as discussed above. Once all information for a station is entered the system redisplays it and offers the opportunity to either accept or reject it. If the information is accepted control passes back to the main menu. Otherwise the prompt list is repeated until the information is accepted.

II.B.2. Load a List from Disk

If there is already a station list on the diskette, it can be loaded and manipulated. The Load selection accomplishes this task. The program asks for the name of a station list file. The user may respond with the name of a file or

may accept the default station list file which has the name STLIST. The default is taken by entering only a carriage return (CR).

II.B.3. Print the List on the Screen

The contents of the list currently in the working memory are reviewed by using the Print selection. The information for each station in the list is displayed on the screen. The program pauses between each station to allow the user a chance to look at the information. Printing is resumed by entering a CR.

II.B.4. Change a Station

Changes to the station list are accomplished with the Change selection. The program first prompts for the name of the station to be changed, and then displays the information for that station. Then a series of prompts is issued for each item of information for the station. At this point the information can either be left as it was or new information can be provided. If the old information is to be retained the response to the prompt is a 0 (numeric zero) and a CR. Otherwise the new information is entered. Note that all new information must conform to the same rules as when adding a station. Once all the changes for a station have been entered the system redisplays all of the station information and offers the opportunity to either accept or reject it. If the information is accepted control passes back to the main menu. Otherwise the prompt list is repeated until the information is accepted.

II.B.5. Delete a Station

Should it become necessary to remove a station from the list, the Delete option is selected. A prompt for the name of the station to be deleted appears. The user responds with a four character station name. If the station is on the list all of the information pertaining to it is displayed and the user is asked to confirm the deletion by replying with the letter "y". Any response other than a "y" ignores the deletion. In either case control passes back to the main menu.

II.B.6. Save the List on Disk

When the desired configuration of the station list has been achieved it must be saved on the diskette by selecting the Save option from the main menu. A prompt appears for the file name to use for the save operation. Valid file names contain up to 10 alphanumeric characters. The default file name, STLIST, can be selected by replying to the prompt with a CR. Once the save operation is completed control returns to the main menu. The user should be aware that the save operation overwrites any file having the name specified for the save.

II.B.7. Quit

The Quit selection on the main menu allows for an orderly exit from the station list maintenance program. Control is returned to the main command level of the P-System.

II.B.8. Initialize a New List

There may be cases in dealing with station lists when the user wishes to start a new list after making changes to one that already exists. This is accomplished by selecting the Initialize option. Any list currently in memory is erased. Note that this does not affect any list already stored on the diskette unless a subsequent Save is performed with an appropriate file name.

III. Polling and Recording

A general description of the polling and recording program, MULTIAC, appears in the System Overview section of this document. What follows is a guide to the operation of the program.

III.A. Program MULTIAC

The polling and recording program is stored on a program diskette under the name SYSTEM.STARTUP. This diskette must be in drive 1. Any time the system is turned on or reset an automatic booting process executes. The P-System looks for a file named SYSTEM.STARTUP during the booting process. If the name is found the program is loaded, and it automatically begins execution. The program assumes that the hardware has been setup as described in the hardware users' guide. There are two diskette files that are required for system operation, and one optional file. The first of the required files is a data recording file on the diskette in drive 2. Refer to Data Diskette Preparation in Section IV.A for the details regarding preparation of diskettes. The other required file is a station list which must be built according to the procedures outlined in Section III above. The optional file is used for recording error information during the polling process. Instructions for preparing this file appear in Section IV.B, Error Log Preparation.

III.A.1. Normal Operation

Program MULTIAC is designed to operate without user intervention unless the user wishes to alter the program's environment. Screen displays during normal operation provide a continuous review of system status. The top line of the monitor contains a status message in the format "STATUS: s DISK nn% FULL ERROR LOG xxx". The "s" holds a status code which is interpreted as follows:

S: system is in the startup process.

W: system is waiting between polling cycles.

P: system is polling a station.

R: system is receiving data from a station.

I: system is decoding the received data.

D: system is recording data on diskette.

The "nn%" indicates how much space on the data diskette has been used. The "xxx" is used to indicate the status of the error logging facility. Codes used are ON, indicating that the error log facility is enabled, and OFF to indicate that the facility is disabled.

When the system status code is W there usually is a station status summary report on the next few lines on the monitor. This report consists of a series of station names in the same order as they appear on the station list. Each name is followed by a number which indicates the number of times the station was polled in its last cycle. A zero here indicates that the station has not been polled since the last system startup. A number from 1 thru 4 indicates the number of tries in order to receive a transmission without errors. Since the

system tries only 5 times for any given station, there is a good chance that any station with a 5 reported was not successfully polled.

The other information appearing on the screen during normal operation consists of the data received from each station as it is polled. There is a line of data for each channel that reports back, showing the station identification number, the channel number and the data value. Following the information for each station is the time of day corresponding to the time at the beginning of the polling cycle. The seconds portion of this time figure reflects the number of tries minus one for the station.

III.A.2. Keyboard Commands

Whenever the system is in the Wait state, i.e., the displayed status code is W, there are several keyboard commands that it can accept. These will be discussed in alphabetical order.

III.A.2.a. Alter Polling Priorities

Altering of the polling periods of a series of stations on the basis of subnetwork identifications is accomplished using the A command. The system prompts for the subnetwork identification number and the new polling period for this subnet. The polling periods for all stations on the list with the corresponding subnetwork identification code are changed to the new period specified. The new period remains in effect until a superceding A command is issued, or until the system is restarted. Changes of a more permanent nature must be made using the station list maintenance program.

III.A.2.b. Immediate Polling of a Station

The I command is used to force the immediate polling of a named station. The system prompts for the station name, and then goes through the complete process of polling, receiving, interpreting, checking for errors and recording on diskette, just as of the station's normal polling time had come up.

III.A.2.c. Quit

The Q command is used to effect an orderly shutdown of the system and return to the P-System main command level. It should be used in preparation for changing the data diskette, maintaining the station list, examining the error log, or any other time when it is necessary to shut the system down.

III.A.2.d. Time of Day Display

The user can request the system to display the current date and time of day by entering the T command. Note that the T command destroys part of the station status display each time it is invoked. The station status display is restored at the conclusion of the next polling cycle.

IV.A.3. Error Handling

During the actual polling process there are several tests performed on the incoming data to ensure validity. These tests include the following:

1. Station time-out. Polled station must respond within 2.25 seconds.
2. Pattern of start and stop bits in received data must be correct.
3. Received station code must be the same as the polled station code.
4. Received channel numbers must start at zero and be consecutively numbered.

If any of these tests fail the station is repolled. If errors are still detected after a total of five tries the system stores the questionable data and goes on to the next station on the list. In any case, all incoming data which cause retries of the polling process are saved in an error log which can be reviewed in an off-line mode.

III.A.4. Error Log

During system startup, the program looks for a file called ERRLOG on the program diskette. This file serves as a scratchpad on which the program can record incoming data which has caused a polling retry. If the file is present on the program diskette then error logging is automatically enabled, and the system status indicates ERROR LOG ON. Recording on the error log starts at the beginning and overwrites previous log information each time the system is restarted. Error logging is disabled automatically if there is no file named ERRLOG present on the program diskette or if the error log file becomes full. There is enough space in the error log to hold 160 entries.

The error log can be inspected with the utility program PLOG. This program and a description of the information in the error log are discussed in Section IV.C.

IV. Utility Functions

This section describes the usage of various utilities that are part of the polling telemetry system. Some of these utilities involve use of UCSD-P System utilities. See Apple Pascal Operating System Reference Manual for the details regarding those utilities.

IV.A. Data Diskette Preparation

Data acquired by the polling system are stored in a large circular buffer file on diskette. A pointer is maintained at the beginning of the file so that the program always knows where the next available diskette record is. When the buffer becomes full, wraparound occurs, and recording resumes at the beginning of the file.

The diskettes used for data storage must be specially prepared before the polling program can successfully operate. Preparation steps include using operating system utilities to format and reserve space on the diskette for the buffer file, and initializing the record pointer at the beginning of the buffer using a special utility program, INSPECTOR.

IV.A.1. Formatting Diskettes

Diskettes must be formatted before the operating system can use them to store data. There is a copy of the formatter program on the polling utilities diskette, POLUTL. Insert this diskette in the drive 1 and boot the system. Give the command X, at which time the system prompts for the name of a file to execute. The proper response is FORMATTER. Insert the diskette to be formatted in drive 2. The formatter program then asks which disk is to be formatted. Reply 5. If the diskette has been previously used, the system asks if it's OK to destroy it. Respond accordingly. When the diskette is formatted the system again asks which disk to format. At this time another diskette can be formatted by inserting it in the drive 2 and replying 5, or the formatter can be terminated by replying Q.

IV.A.2. Creating the Data File

This step reserves the entire diskette for data recording. Do this by invoking the filer with the command F. Then give the M command to make a file. The file to make and its size are specified as BLANK:DATA[274].

IV.A.3. Program INSPECTOR

Now that a diskette has been prepared and file space reserved, the record pointer at the beginning of the file has to be set. This is done using program INSPECTOR. This program has several other functions which will also be discussed.

Invoke INSPECTOR from the main command level by giving the command X and asking for INSPECTOR. The following menu appears.

R(eposition end of file

S(eek and read

Z(ero out first record

Q(uit

The selection to initialize the pointer at the beginning of the file is Z. Once this is done, leave the program via the Q command.

This program also allows you to look at any record on the diskette. Use the seek and read command to do this. The file holds 4,384 records, numbered 0 to 4,383. The seek command prompts for the number of the record to be sought, then reads the record from the diskette and displays it on the screen. Seeking record zero shows the current end of file pointer, also referred to as the last record number. It is the first number displayed as a result of the seek command.

Notice that seek displays two lines of numbers on the screen. The first line contains the following: year, month, day, hour, minute, second, received station number, and a number which is a combination of the number of channels and the polled station number for this entry. Recover the number of channels by dividing by 256. The remainder is the polled station number. For example, the number 1795 means 1795/256 = 7 channels, and 1795 - (256*7) = 3, meaning station number 3 was polled. The second line of eight numbers are the data values for the data channels. Only as many of these numbers as there are data channels for a particular station will mean anything.

The logical end of the recording file can be reset to any desired position by using the R command. It prompts for the desired number of last record, and sticks this number in the pointer at the beginning of the file. This can be very useful if the data diskette has accidentally had the end of file reset by the Z command. Note that the Z command does a logical erase of the diskette, but does not actually erase the data stored on the diskette.

IV.B. Error Log File Preparation

Program MULTIAC records errors that cause polling retries if there is a file ERRLOG on the program diskette. This file can be created from the main command level by invoking the filer (use the F command to do this) and making a file (with the M command) with the specification ERRLOG[20]. Then quit the filer. All that is required is a minimum of 20 blocks of space on the program diskette.

IV.C. Program PLOG - Review Error Log

The program PLOG presents the contents of the error log file on the computer screen for review. All items displayed are plainly labeled. Entries are displayed one at a time, and the program waits for you to hit a key on the keyboard before continuing. You can hit a Q to quit the program.

Of the information displayed by PLOG, the channel column requires the most interpretation. The values displayed in this column normally show the received channel number for the data, and they should be in the range 0 thru 7. If a framing error was detected in the incoming data (expected hi and lo bits were not in the correct state), the received channel has the value 128 added.

IV.D. Program PDATA - Review Received Data

This program allows for graphic review of data recorded on the data diskette. Invoke it from the main command level with the command X. The file to execute is PDATA. A series of self explanatory questions appears. Supply information according to what data you wish to review. Once you have supplied all the answers, PLOG reads through the data file looking for data that match your specifications. A maximum of 450 entries are then stored, each including time and up to 8 data values corresponding to the eight data channels. Once the data are loaded, the program asks which parameter you wish to display. Answer 0 to exit the program, or 1 through 8 depending on which data channel you wish to see. Then supply the minimum and maximum Y values for the screen plot. The program draws the plot, then waits for you to hit the return key. It then gives you a chance to look at additional channels.

IV.E. Program CLOCK - Set the System Clock

The system clock is set with the program CLOCK. All of the program prompts are self explanatory. The clock should be set when the system is initially started. Resetting is performed as necessary to correct for drift. The clock has batteries to keep it running when the system is shut off. Checking the current time is accomplished via the T command in the polling program.

APPENDIX: Program Source Listings

```
Program CLOCK;

Var
  date: array [0..15] of integer;
  i: integer;

Procedure SetClock;
External;

Function GetInt (prompt: string): integer;
Var ijunk, jjunk: integer;
    cjunk: char;
Begin
  Repeat
    Write (prompt, '->');
    (*$I-*)
    Read (ijunk);
    jjunk := ioresult;
    (*$I+*)
    If jjunk <> 0 then begin
      Read (cjunk);
      Writeln ('Bad input, try again.', chr (7))
    End (* IF *)
  Until jjunk = 0;
  Read (cjunk);
  Getint := ijunk
End; (* GetInt *)

Begin
  date[0] := GetInt ('Year (YY)');
  date[1] := date[0] mod 10;
  date[0] := date[0] div 10;
  date[2] := GetInt ('Month (MM)');
  date[3] := date[2] mod 10;
  date[2] := date[2] div 10;
  date[4] := GetInt ('Day (DD)');
  date[5] := date[4] mod 10;
  date[4] := date[4] div 10;
  date[7] := GetInt ('Hour (HH)');
  date[8] := date[7] mod 10;
  date[7] := (date[7] div 10) + 8;
  date[9] := GetInt ('Minute (MM)');
  date[10] := date[9] mod 10;
  date[9] := date[9] div 10;
  Setclock
End.
```

```
Program Inspector;
Type
  stationdata = record
    year, month, day, hour, minute, second, stnum, nchan: integer;
    dvalues: array [0..7] of integer
  End;

Var
  i,j, nrec: integer;
  answ:char;
  batches: file of stationdata;

Procedure seek_and_read;
Begin
  Write ('Seek which record? '); Readln (i);
  Seek (batches, i);
  Get (batches);
  If eof(batches) then
    Begin
      Writeln ('That is at or beyond the end of file');
      Exit (seek_and_read_)
    End;
  With batches^ do
    Begin
      Write (year:5, month:3, day:3, hour:3, minute:3);
      Writeln (second:3, stnum:5, nchan:2);
      For j := 0 to 7 do Write (dvalues[j]:5)
    End;
  Writeln
End;

Procedure quit;
Begin
  Close (batches);
  Exit (program)
End;

Procedure zero_rec;
Begin
  Seek (batches, 0);
  batches^.year := 0;
  Put (batches)
End;

Procedure rset_rec;
Begin
  Write ('Desired number of last record? ');
  Readln (nrec);
  Seek (batches, 0);
  batches^.year := nrec;
  Put (batches)
End;

Begin
  Reset (batches,'blank:data');
  Repeat
    Writeln ('R(eposition end of file');
    Writeln ('S(eek and read');
```

```
    Writeln ('Z(ero first record');
    Write ('Q(uit ->');read(answ);
    Writeln;
    Case answ of
      'q': quit;
      'Q': quit;
      'r': rset_rec;
      'R': rset_rec;
      's': seek_and_read;
      'S': seek_and_read;
      'z': zero_rec;
      'Z': zero_rec
    End;
  Until false
End.
```

```
(*$S+*)
Program Multiac;
  Uses Applestuff;

Const
  Maxsta = 63;

Type
  StationData = Record
    year, month, day, hour, minute, second, stnum, stchan: integer;
    values: array[0..7] of integer;
  end;

  Strecord = Record
    Stid, Stfreq, Sttc1, Sttc2, Sttc3, Stper, Stsid, Stachan: integer;
    Stname: String[4];
    Stmux, stzvolts, stbatc, stbatt: integer;
  end;

  ErrRec = Record
    Epsta, Eyr, Emo, Eda, Ehr, Emin, Esec, Epdum: integer;
    Esta, Echan, Eval: array [0..7] of integer;
  end;

  minutes = 0..60;
  periodset = set of minutes;

Var
  Stalist: array[0..Maxsta] of Strecord;
  Stfile: file of strecord;
  Batches: file of StationData;
  ErrLog: file of ErrRec;
  station, channel, value: array[0..7] of integer;
  date: array[0..15] of integer;
  LastRecord, Nsta: integer;
  prot: integer;
  delay, intrv, period, nchan, errcnt: integer;
  trial: array[0..7] of integer;
  ntries: array [0..MaxSta] of integer;
  ThisPeriod, ValidTime: periodset;
  keyin: char;
  Polled, ErrStatus, Novax: boolean;
  Ascline, Intstring: String;

Procedure Receive(delay, intrv: integer);
External;

Procedure Decode;
External;

Procedure HoldClock;
External;

Procedure ReadClock;
External;

Procedure Poll(freq, id1, id2, id3: integer);
External;
```

```
Procedure Tinit;
External;

Procedure Send3 (Aline: String; i: Integer);
External;

Procedure ConnectToVax;
Begin
  Ascline := ''; Prot := 0; Send3 (Ascline, Prot);
  For Prot := 0 to 1500 do; Prot := 0;
  Ascline := 'SESAME'; Send3 (Ascline, Prot);
  Novax := true
End;

Procedure StartVax;
Begin
  Ascline := 'START'; Prot := 0;
  Send3 (Ascline, Prot);
  Novax := false
End;

Procedure StopVax;
Begin
  Ascline := 'STOP'; Prot := 0;
  Send3 (Ascline, Prot);
  Novax := true
End;

Procedure ShowStatus (Status: char);
Var
  i: integer;
Begin
  GoToXY (0, 0);
  i := LastRecord div 44;
  Write ('Status: ', Status, ' Disk', i:3, '% full Error Log ');
  If ErrStatus then Writeln ('ON') else Writeln ('OFF');
  GoToXY (0, 23)
End;

Procedure FindEnd;
Begin
  Seek (Batches, 0);
  Get (Batches);
  LastRecord := Batches^.Year;
  Seek (Batches, LastRecord);
  Get (Batches)
End;

Procedure PrepareDisk;
Var
  GoodFile: Boolean;
  i: integer;
Begin
  GoodFile := false;
  Repeat
    (*$I-*)
    Reset (Batches, 'blank:data');
```

```
    i := IOResult;
    (*$I+*)
    If i = 0 then
      Begin
        FindEnd;
        GoodFile := true
      End;
    If (i = 9) or (i = 10) then
      Begin
        Write ('Q(uit or N(ew disk? ');
        Readln (keyin);
        If (keyin = 'q') or (keyin = 'Q') then exit (program);
        Writeln ('Please put in a data diskette.');
        Writeln ('Then hit RETURN');
        Readln (keyin)
      End;
    If (i <> 0) and (i <> 9) and (i <> 10) then
      Begin
        Writeln ('Unknown problem in procedure PrepareDisk');
        Writeln ('Error code is ', i);
        Exit (program)
      End
  Until GoodFile
End;

Procedure PrepError;
Var
  i: integer;
Begin
  (*$I-*)
  Reset (ErrLog, 'Apple1:errlog');
  i := ioresult;
  (*$I+*)
  If i = 0 then
    Begin
      ErrStatus := true;
      errcnt := 0;
      Seek (ErrLog, 0)
    End;
  If i <> 0 then ErrStatus := false
End;

Procedure InitTrial;
Begin
  Delay := 2030; Intrv := 1675;
  Trial[0] := 60;  Trial[1] := 30;  Trial[2] := 20;
  Trial[3] := 15;  Trial[4] := 10;  Trial[5] := 5;
  Trial[6] := 2;   Trial[7] := 1;
  ValidTime := [1, 2, 5, 10, 15, 20, 30, 60]
End;

Procedure LoadStationList;
Begin
  Reset (Stfile, 'Stlist');
  Nsta := 0;
  While (not (eof(Stfile))) and (Nsta <= Maxsta) do
    Begin
      Stalist[Nsta] := Stfile^;
```

```
      ntries[Nsta] := 0;
      Nsta := Succ (Nsta);
      Get (Stfile)
    End;
  Nsta := Pred (Nsta);
  Close (Stfile)
End;

Procedure AlterPriority;
Var
  I, pnum, nper: integer;
Begin
  Write ('Enter Subnet ID '); Readln (pnum);
  Write ('Enter new polling period '); Readln (nper);
  If not (nper in ValidTime) then
    Begin
      Writeln ('Not a valid polling period');
      Exit (AlterPriority)
    End;
  For i := 0 to nsta do
    Begin
      If Stalist[i].stsid = pnum then
        Begin
          Stalist[i].stper := nper;
          Writeln (Stalist[i].stname, ' changed.')
        End
    End;
  Writeln (' ')
End;

Procedure StaStatus;
Var
  i: integer;
Begin
  GoToXY (0, 1);
  Write (chr(29));
  For i := 0 to nsta do
    Begin
      Write (stalist[i].stname, ntries[i]:2, ' ');
      If (((i + 1) div 5) * 5) = (i + 1) then
        Begin
          Writeln(' ');
          Write (chr(29))
        End
    End;
  Writeln (' '); Writeln (chr(29)); GoToXY (0,23)
End;

Procedure FindPeriod;
Var
  mins, test, i: integer;
Begin
  Mins := (date[3] * 10) + date[2];
  If Mins = 0 then Mins := 60;
  For i := 0 to 7 do
    Begin
      Period := Trial[i];
      Test := (Mins div Period) * Period;
```

```
      If Test = Mins then Exit (FindPeriod)
    End
End;

Procedure Report;
Var
  i: integer;
Begin
  For i := 0 to nchan do
    Writeln (station[i]:2, ' ',channel[i]:3,' ',value[i]:4)
End;

Procedure TwoOut(item: integer);
Begin
  Write (Date[item]:1, Date[item-1]:1)
End;

Procedure ShowTime;
Begin
  TwoOut(10);Write('/');TwoOut(8);Write('/');TwoOut(12);Write(' ');
  Twoout(5); Write(':');TwoOut(3);Write(':');TwoOut(1); Writeln(' ')
End;

Procedure LogError (n: integer);
Var
  i: integer;
Begin
  If (not ErrStatus) then Exit (LogError);
  With ErrLog^ do
    Begin
      Emo := (date[10] * 10) + date[9];
      Eda := (date[8] * 10) + date[7];
      Eyr := (date[12] * 10) + date[11];
      Ehr := (date[5] * 10) + date[4];
      Emin := (date[3] * 10) + date[2];
      Esec := (date[1] * 10) + date[0];
      Epsta := n;
      Epdum := 0;
      For i := 0 to 7 do
        Begin
          Esta[i] := station[i];
          Echan[i] := channel[i];
          Eval[i] := value[i]
        End
      End;
    errcnt := succ (errcnt);
    (*$I-*)
    Put (ErrLog);
    i := ioresult;
    (*$I+*)
    If (i <> 0) or (errcnt > 319) then
      Begin
        Close (ErrLog);
        ErrStatus := false
      End
End;

Procedure Save (n: integer);
```

```
Var
  i: integer;
Begin
  ShowStatus ('d');
  If LastRecord > 4381 then begin
    LastRecord := 1;
    Seek (Batches, LastRecord);
    Get (Batches)
    End;
  With Batches^ do
    Begin
      Month := (date[10] * 10) + date[9];
      Day := (date[8] * 10) + date[7];
      Year := (date[12] * 10) + date[11];
      Hour := (date[5] * 10) + date[4];
      Minute := (date[3] *10) + date[2];
      Second := (date[1] *10) + date[0];
      Stnum := station[0];
      Stchan := (nchan * 256) + n;
      Ascline := '1';
      Str (Year, Intstring); Ascline := Concat (Ascline, ' ', Intstring);
      Str (Month, Intstring); Ascline := Concat (Ascline, ' ', Intstring);
      Str (Day, Intstring); Ascline := Concat (Ascline, ' ', Intstring);
      Str (Hour, Intstring); Ascline := Concat (Ascline, ' ', Intstring);
      Str (Minute, Intstring); Ascline := Concat (Ascline,' ',Intstring);
      Str (Second, Intstring); Ascline := Concat (Ascline,' ',Intstring);
      Str (Stnum, Intstring); Ascline := Concat (Ascline, ' ', Intstring);
      Str (Stchan, Intstring); Ascline := Concat (Ascline, ' ', Intstring);
      Prot := 0; Send3 (Ascline, Prot);
      Ascline := '2';
      For i := 0 to 7 do begin
        values[i] := value[i];
        Str (value[i], intstring);
        Ascline := Concat (Ascline, ' ', Intstring)
      End;
      Send3 (Ascline, Prot)
    End;
  Put (Batches);
  LastRecord := succ (LastRecord)
End;

Procedure PollAndRecord (ista: integer);
Var
  i, freq, id1, id2, id3: integer;
  bad: boolean;
Begin
  Bad := true;
  With StaList[ista] do
    Begin
      freq := stfreq;
      id1 := sttc1;
      id2 := sttc2;
      id3 := sttc3;
      nchan := stachan
    End;
  Ntries[ista] := 0;
  Repeat
    Begin
```

```
      For i := 0 to 7 do
        Begin
          station[i] := 0;
          channel[i] := 0;
          value[i] := 0
        End;
      ShowStatus ('p');
      Poll (freq, id1, id2, id3);
      ShowStatus ('r');
      Receive (Delay, Intrv);
      ShowStatus ('i');
      Decode;
      Report;
      Showtime;
      Ntries[ista] := succ (Ntries[ista]);
      date[0] := Ntries[ista];
      Bad := false;
      For i := 0 to nchan do
        Begin
          If station[i] <> Stalist[ista].stid then Bad := true;
          If channel[i] <> i then Bad := true
        End;
      If Bad then LogError (Stalist[ista].stid);
      If Ntries[ista] > 4 then Bad := false
    End
  Until (not Bad);
  Save (Stalist[ista].stid);
  Polled := true
End;

Procedure Checksta;
Var
  i: integer;
Begin
  Polled := false;
  For i := 0 to nsta do begin
    If Stalist[i].stper in ThisPeriod then begin
      If Novax then StartVax;
      PollAndRecord(i)
    End
  End;
  If (not Polled) then exit (Checksta);
  Seek (Batches, 0);
  Batches^.year := LastRecord;
  Put (Batches);
  Seek (Batches, LastRecord);
  Get (Batches);
  StopVax;
  ShowStatus ('w');
  StaStatus
End;

Procedure CheckTime;
Begin
  HoldClock;
  ReadClock;
  If (date[0] <> 0) or (date[1] <> 0) then Exit (CheckTime);
  FindPeriod;
```

```
  Case Period of
    1: ThisPeriod := [1];
    2: ThisPeriod := [1, 2];
    5: ThisPeriod := [1, 5];
    10: ThisPeriod := [1, 2, 5, 10];
    15: ThisPeriod := [1, 5, 15];
    20: ThisPeriod := [1, 2, 5, 10, 20];
    30: ThisPeriod := [1, 2, 5, 10, 15, 30];
    60: ThisPeriod := [1, 2, 5, 10, 15, 20, 30, 60]
  End;
  Checksta
End;

Procedure Quit;
Begin
 Close (batches);
 Close (ErrLog);
 Exit (program)
End;

Procedure ShowCurrentTime;
Begin
  HoldClock;
  ReadClock;
  ShowTime;
  ShowStatus ('W')
End;

Procedure PollImmediate;
Var
  i: integer;
  iname: string[4];
Begin
  ShowCurrentTime;
  Write ('Name of station to poll? ');readln (iname);
  For i := 0 to nsta do
    Begin
      If Stalist[i].stname = iname then
        Begin
          StartVax;
          PollAndRecord (i);
          Seek (Batches, 0);
          Batches^.year := LastRecord;
          Put (Batches);
          Seek (Batches, LastRecord);
          Get (Batches);
          ShowStatus ('w');
          StopVax;
          StaStatus
        End
      End
  End;

  Begin
  PrepareDisk;
  PrepError;
  ShowStatus ('S');
  InitTrial;
```

```
  TInit;
  ConnectToVax;
  LoadStationList;
  ShowCurrentTime;
  Repeat
    Repeat
      CheckTime
    Until keypress;
    Read (keyin);
    Writeln;
    Case keyin of
      'a': AlterPriority;
      'A': AlterPriority;
      'i': PollImmediate;
      'I': PollImmediate;
      'q': Quit;
      'Q': Quit;
      't': ShowCurrentTime;
      'T': ShowCurrentTime
    End
  Until false
End.
```

```
Program PData;
  Uses TurtleGraphics;

Const
  MaxSta = 63;
  MaxEnt = 449;

Type
  StationData = Record
    year, month, day, hour, minute, second, stnum, stchan: integer;
    values: array[0..7] of integer;
  end;

  StRecord = Record
    stid, Stfreq, Sttc1, Sttc2, Sttc3, Stper, Stsid, Stachan: integer;
    Stname: String[4];
    Stmux, Stzvolts, Stbatc, Stbatt: integer;
  end;

  StationTab = Record
    Stid: integer;
    Stname: String[4];
  end;

Var
  StaTab: array[0..MaxSta] of StationTab;
  Time: array[0..MaxEnt] of integer;
  DValue: array[0..MaxEnt, 0..7] of integer;
  fy, nent, ipar, nsta, sta, nchan, psta, i, j, y: integer;
  nrec, lastrec: integer;
  mny, mxy: integer;
  bday, cday, eday: real;
  xrat, yrat: real;
  ch: char; StTry: String; Good: Boolean;
  StFile: file of StRecord;
  Batches: file of StationData;

Function Dayjl (Year, Month, Day, Hour, Minute: Integer): Real;
Var
  T1, T2: integer;
  T3, T4: real;
Begin
  T2 := Day;
  T1 := (Year div 4) * 4;
  If (T1 = Year) and (Month > 2) then T2 := T2 + 1;
  Case Month of
    1:   T1 := 0;    2: T1 := 31;   3: T1 := 59;
    4:   T1 := 90;   5: T1 := 120;  6: T1 := 151;
    7:   T1 := 181;  8: T1 := 212;  9: T1 := 243;
    10:  T1 := 273; 11: T1 := 304; 12: T1 := 334
  End;
  T3 := Hour / 24.0;
  T3 := T3 + (Minute / 1440.0);
  If Year = fy then T4 := 0.0 else T4 := 365.0;
  Dayjl := T1 + T2 + T3
End;

Function GetInt (prompt: string): integer;
```

```
Var ijunk, jjunk: integer;
    cjunk: char;
Begin
  Repeat
    Write (prompt, '->');
    (*$I-*)
    Read (ijunk);
    jjunk := ioresult;
    (*$I+*);
    If jjunk <> 0 then begin
      Read (cjunk);
      Writeln ('Bad input, try again.', chr (7))
    End (* IF *)
  Until jjunk = 0;
  Read (cjunk);
  GetInt := ijunk
End; (* GetInt *)

Procedure LoadStationList;
Begin
  Reset (StFile, 'STLIST');
  Nsta := 0;
  While (not eof(StFile)) and (nsta <= MaxSta) do begin
    StaTab[nsta].Stid := StFile^.Stid;
    StaTab[nsta].Stname := StFile^.Stname;
    Nsta := succ(Nsta);
    Get (StFile)
  End; (* While *)
  Nsta := pred (Nsta);
  Close (StFile)
End; (* LoadStationList *)

Procedure GetStation;
Begin
  Good := false;
  Repeat begin
    Write ('Name of Station to Plot? '); Readln (StTry);
    If length (StTry) = 4 then begin
      For i := 0 to nsta do begin
        If StaTab[i].stname = StTry then begin
          sta := StaTab[i].Stid;
          Good := true
        End (* If *)
      End (* For *)
    End (* If *)
  End; (* Repeat *)
  Until Good
End; (* GetStation *)

Procedure LoadData;
var
  mm, dd, yy, hh, mi: integer;
Begin
  Repeat
    mm := GetInt ('Beginning Date - MM');
    dd := GetInt ('DD');
    yy := GetInt ('YY');
    hh := GetInt ('Beginning Time - HH');
```

```
    mi := GetInt ('MM');
    Fy := yy; bday := Dayjl (yy, mm, dd, hh, mi);
    mm := GetInt ('Ending Date - MM');
    dd := GetInt ('DD');
    yy := GetInt ('YY');
    hh := GetInt ('Ending Time - HH');
    mi := GetInt ('MM');
    eday := Dayjl(yy, mm, dd, hh, mi);
    Write ('These values OK?'); Read (ch)
  Until ((ch = 'y') or (ch = 'Y'));
  xrat := 280.0/(eday - bday);
  Reset (batches, 'blank:data');
  LastRec := Batches^.year;
  Get (Batches);
  Nent := 0;
  Good := true;
  Nrec := 1;
  While (Nrec <= Lastrec) and Good and (Nent < MaxEnt) do begin
    With batches^ do begin
      nchan := stchan div 256;
      psta := stchan - (nchan * 256);
      cday := dayjl(year, month, day, hour, minute);
      If (Stnum = Sta) and (psta = Sta) and (cday >= bday) then begin
        If cday > eday then Good := false;
        Time[Nent] := round ((cday - bday) * xrat);
        For i := 0 to 7 do DValue[Nent, i] := values[i];
        Nent := succ (Nent)
      End (* If *)
    End; (* With *)
    Get (Batches);
    Nrec := succ (Nrec)
  End; (* While *)
  Nent := pred (Nent)
End; (* LoadData *)

Procedure GetParam;
Begin
  ipar := GetInt ('Parameter to plot, 0 to quit');
  ipar := ipar - 1;
  If ipar < 0 then exit (program);
  mny := GetInt ('Minimum Y value');
  mxy := GetInt ('Maximum Y value');
  yrat := 192.0/(mxy - mny)
End; (* GetParam *)

Procedure XLines (size: integer);
Begin
  i := round ((cday - bday) * xrat);
  PenColor (white);
  MoveTo (i, 0); MoveTo (i, size);
  PenColor (none); MoveTo (j, 191);
  PenColor (white);
  MoveTo (i, 191); MoveTo (i, 191 - size);
  j := i; PenColor (none); MoveTo (i, 0)
End; (* XLines *)

Procedure YLines (size: integer);
Begin
```

```
  i := Round ((y - mny) * yrat);
  PenColor (white);
  MoveTo (0, i); MoveTo (size, i);
  PenColor (none); MoveTo (279, j);
  PenColor (white);
  MoveTo (279, i); MoveTo (279 - size, i);
  j := i; Pencolor (none); MoveTo (0, i)
End; (* YLines *)

Procedure DrawBox;
Begin
  cday := trunc (bday);
  PenColor (none); MoveTo (0, 0);
  j := 0;
  Repeat
    cday := cday + 0.5;
    XLines (2);
    cday := cday + 0.5;
    XLines (4)
  Until cday > eday;
  MoveTo (0, 0);
  y := (mny div 10) * 10;
  j := 0;
  Repeat
    y := y + 10;
    If ((y div 100) * 100) = y then Ylines (4) else YLines (2)
  Until y > mxy;
  MoveTo (0, 0)
End; (* DrawBox *)

Procedure PlotData;
Var
  dummy: string;
Begin
  Grafmode; FillScreen (black);
  DrawBox;
  For i := 0 to Nent do begin
    y := round ((abs(DValue[i, ipar]) - mny) * yrat);
    MoveTo (Time[i], y);
    PenColor (white)
  End; (* For *)
  Readln (dummy)
End; (* PlotData *)

Begin
  LoadStationList;
  GetStation;
  LoadData;
  If nent <= 0 then begin
    Writeln ('No data found for that station');
    exit (program)
  End; (* If *)
  InitTurtle;
  Repeat
    Textmode;
    Repeat
      GetParam;
      Write ('These values OK?'); Read (ch)
```

```
    Until ((ch = 'y') or (ch = 'Y'));
    PlotData
  Until false
End. (* PData *)
```

```
Program PLog;
  Uses AppleStuff;

Type
  ErrRec = Record
    Epsta, Eyr, Emo, Eda, Ehr, Emin, Esec, Edum: integer;
    Esta, Echan, Eval: array [0..7] of integer;
  End;

Var
  ErrLog: file of ErrRec;
  i: integer;
  keyin: char;

Begin
  Reset (ErrLog, 'Apple1:errlog');
  Repeat
    Begin
      With ErrLog^ do
      Begin
        Writeln ('Polled Station ID: ', Epsta:4);
        Writeln ('Date and Time: ', Emo:2,'/',Eda:2,'/',Eyr:2,' ',
                 Ehr:2,':',Emin:2,':',Esec:2);
        Writeln ('Sta.  Chan.  Value');
        For i := 0 to 7 do Writeln (Esta[i]:4, Echan[i]:7, Eval[i]:7);
        Repeat i:= 0 until KeyPress;
        Read (keyin)
      End;
      Get (ErrLog)
    End
  Until ( eof (ErrLog) or (keyin = 'Q'));
  Close (ErrLog)
End.
```

```
Program Stationlist;

Const
  Maxsta = 127;

Type
  Minutes = 0..59;
  Timeset = set of Minutes;
  Stlist = record
    Stid, Stfreq, Sttc1, Sttc2, Sttc3, Stper, Stsid, Stchan: integer;
    Stname: string[4];
    Stmux, Stzvolts, Stbatc, Stbatt: integer;
  end;

Var
  Stalist: array[0..Maxsta] of Stlist;
  Stfile: file of Stlist;
  ista, i, j, nsta: integer;
  fname: string;
  comm: char;
  EmptyList: boolean;

Function Tran(id: integer): integer;
Begin
  tran := 0;
  case id of
    0: tran := 40;  1: tran := 17;  2: tran := 33;
    3: tran := 65;  4: tran := 18;  5: tran := 34;
    6: tran := 66;  7: tran := 20;  8: tran := 36;
    9: tran := 68
  end (* case *)
End; (* Tran *)

Procedure Sttcget(ind: integer);
Begin
  With Stalist[ind] do
  Begin
    sttc1 := stid div 100;
    sttc2 := (stid - (sttc1 * 100)) div 10;
    sttc3 := stid - (sttc1 * 100) - (sttc2 * 10);
    sttc1 := tran(sttc1);
    sttc2 := tran(sttc2);
    sttc3 := tran(sttc3)
  End (* with *)
End; (*sttcget *)

Procedure Stdisp(Ind: integer);
Var
  atchan: integer;
Begin
  With Stalist[ind] do
  Begin
    Writeln ('Station name: ', stname);
    Writeln ('Station ID:   ', stid);
    Writeln ('Radio Frequency is ', stfreq);
    Writeln ('Tone codes are ', sttc1, ' ', sttc2, ' ', sttc3);
    Writeln ('Polling period is ', stper, ' minutes.');
    Writeln ('Subnet ID is  ', stsid);
```

```
    atchan := stchan +1;
    Writeln ('Number of channels is ', atchan);
    Writeln ('Mux Serial Number is ', stmux);
    Writeln ('Bit count for zero volts is ', stzvolts);
    Writeln ('Battery conversion constant is ', stbatc);
    Writeln ('Battery status flag is ', stbatt)
  End (* with *)
End; (* Stdisp *)

Procedure Stpack;
Begin
  If nsta = 0 then
  Begin
    nsta := -1;
    emptylist := true;
    exit (stpack)
  End; (* If *)
  For ista := i to (nsta - 1) do
    Stalist[ista] := Stalist[ista + 1];
  nsta := nsta - 1
End; (* Stpack *)

Procedure Stadd;
Begin
  i := nsta + 1;
  If i > MaxSta then
  Begin
    Writeln ('The station list is full!');
    Exit (Stadd)
  End; (* If *)
  With Stalist[i] do
  Repeat
    (*$I-*)
    Write ('Four letter station name? '); Readln (stname);
    Write ('Station id? '); Readln (stid);
    Write ('Radio frequency? '); Readln (stfreq);
    Writeln ('Polling period');
    Write ('(1,2,5,10,15,30,60 minutes)? '); Readln (stper);
    Write ('Subnet ID? '); Readln (stsid);
    Write ('Number of channels? '); Readln (stchan);
    stchan := stchan - 1;
    Write ('Mux serial number? '); Readln (stmux);
    Write ('Zero volts bit count? '); Readln (stzvolts);
    Write ('Battery conversion constant? '); Readln (stbatc);
    Write ('Battery status flag? '); Readln (stbatt);
    (*$I+*)
    Sttcget(i);
    Write (chr(12)); GotoXY (0,0);
    Stdisp(i);
    Write ('Are these values correct? '); Read (comm)
  Until ((comm = 'y') or (comm = 'Y'));
  nsta := i;
  emptylist := false
End; (* Stadd *)

Procedure Stchange;
Var
  chname: string[4];
```

```
Begin
  If EmptyList then Exit (Stchange);
  Write ('Name of station to change? '); Readln (chname);
  For i := 0 to nsta do
  Begin
    If stalist[i].stname = chname then
    Begin
      stdisp (i);
      Repeat
        With stalist[i] do
        Begin
          (*$I-*)
          Write ('New station name? '); Readln (chname);
          If Length (chname) = 4 then stname := chname;
          Write ('New station id? '); Readln (j);
          If j <> 0 then stid := j;
          Write ('New radio frequency? '); Readln (j);
          If j <> 0 then stfreq := j;
          Write ('New polling period? '); Readln (j);
          If j <> 0 then stper := j;
          Write ('New subnet ID? '); Readln (j);
          If j <> 0 then stsid := j;
          Write ('New number of channels? '); Readln (j);
          If j <> 0 then stchan := j - 1;
          Write ('New mux serial number? '); Readln (j);
          If j <> 0 then stmux := j;
          Write ('New zero volts level? '); Readln (j);
          If j <> 0 then stzvolts := j;
          Write ('New battery conversion constant? '); Readln (j);
          If j <> 0 then stbatc := j;
          Write ('New battery status flag? '); Readln (j);
          If j <> 0 then stbatt := j;
          Writeln
          (*$I+*)
        End; (* with *)
        sttcget (i);
        stdisp (i);
        Write ('If changes are OK reply Y '); Read (comm)
      Until ((comm = 'y') or (comm = 'Y'));
      Exit (stchange)
    End (* If *)
  End (* For *)
End; (* Stchange *)

Procedure Stdelete;
Var
  Delname: string[4];
  Hit: Boolean;
Begin
  If EmptyList then Exit (Stdelete);
  Write ('Name of station to delete? '); Readln (delname);
  hit := false;
  For i := 0 to nsta do
  Begin
    If stalist[i].stname = delname then
    Begin
      stdisp(i);
      Write ('To delete reply Y '); Read (comm);
```

```
      If not ((comm = 'y') or (comm = 'Y')) then Exit (Stdelete);
      stpack;
      hit := true
    End (* If *)
  End; (* For *)
  If (not hit) then Writeln (delname, ': station not on list');
End; (* Stdelete *)

Procedure Stinit;
Begin
  Emptylist := true;
  Nsta := -1
End; (* Stinit *)

Procedure Stload;
Begin
  Repeat
  Begin
    Write ('Name of file to load? '); Readln (fname);
    If Length (fname) = 0 then fname := 'STLIST';
    (*$I-*)
    Reset (stfile, fname);
    i := ioresult;
    (*$I+*)
    If i > 0 then writeln ('Bad file name')
  End
  Until i = 0;
  i := -1;
  While (not eof(stfile)) do
  Begin
    i := i + 1;
    stalist[i] := stfile^;
    get (stfile)
  End; (* while *)
  Close (stfile);
  nsta := i;
  emptylist := false
End; (* Stload *)

Procedure Stprint;
Begin
  If emptylist then exit(stprint);
  For i := 0 to nsta do
  begin
    stdisp(i);
    write('Return to continue'); readln
  end
End; (* Stprint *)

Procedure Stsave;
Begin
  If emptylist then exit(stsave);
  Repeat
    Begin
      Write ('Name of file to save? '); readln (fname);
      If Length (fname) = 0 then fname := 'STLIST';
      (*$I-*)
      ReWrite (Stfile, fname);
```

```
      i := ioresult;
      (*$I+*)
      If i <> 0 then Writeln ('Bad file name')
    End
  Until i = 0;
  For i := 0 to nsta do
  Begin
    Stfile^ := stalist[i];
    Put (stfile)
  End; (* do *)
  Close (stfile, lock)
End; (* Stsave *)

Begin
  Stinit;
  Repeat
    (* put up the menu *)
    Write (chr(12)); (* CTRL-L to clear screen *)
    GotoXY (0,0);
    Writeln ('A(dd a new station');
    Writeln ('C(hange a station');
    Writeln ('D(elete a station');
    Writeln ('I(nitialize a new list');
    Writeln ('L(oad list from disk');
    Writeln ('P(rint list on screen');
    Writeln ('Q(uit');
    Writeln ('S(ave list on disk');
    Write ('Your choice? ');
    Read (comm); Writeln;
    Case comm of
      'A': Stadd;
      'a': Stadd;
      'C': Stchange;
      'c': Stchange;
      'D': Stdelete;
      'd': Stdelete;
      'I': Stinit;
      'i': Stinit;
      'L': Stload;
      'l': Stload;
      'P': Stprint;
      'p': Stprint;
      'Q': Exit(program);
      'q': Exit(program);
      'S': Stsave;
      's': Stsave;
    End (* case *)
  Until false
End.
```

```
        .include poppsh.text
        .proc   decode
        .public station,channel,value
;
; Procedure decode;
;
; Requires var station array[0..7] of integer;
;             channel array[0..7] of integer;
;             value   array[0..7] of integer;
;
; Multi channel decoder program for Apple Pascal system.
;
; Purpose is to decode data received by proc receiver so
; Pascal can deal with it.
;
; Tom English   HVO   July, 1983
;
; Page Zero equates
;
save    .equ    0       ;return addr save area
error   .equ    02      ;error flag
base    .equ    04      ;base addr for buffer
tempx   .equ    06      ;temporary x reg save area
temp    .equ    07
work    .equ    01F00   ;receive buffer
;
        pop     save    ;save return addr
        lda     #00
        sta     base
        lda     #01F
        sta     base+1
        ldy     #0
        ldx     #0
        stx     temp
        stx     tempx
;
rept    sty     error   ;clear error flag
        lda     @base,y ;bit 0 = 0 means no data here
        beq     pau     ;so escape
        iny
        lda     @base,y ;check bits 1,13,14,29,30
        bpl     err     ;for hi
        ldy     #13.
        lda     @base,y
        bpl     err
        iny
        lda     @base,y
        bpl     err
        iny
        lda     @base,y ;bits 15 and 16
        bmi     err     ;must be low
        iny
        lda     @base,y
        bmi     err
        ldy     #29.
        lda     @base,y
        bpl     err
        iny
```

```
        lda     @base,y
        bpl     err
;
; Passed the test
;
ok      ldy     #9.
        ldx     #8.
        jsr     rght8   ;fetch station
        pha             ;save it
        ldy     #12.
        ldx     #3.
        jsr     rght8   ;fetch channel
        ora     error   ;pick up error bit
        pha             ;save it too
        ldy     #28.
        ldx     #4.
        jsr     rght8   ;fetch value hi
        pha
        ldy     #24.
        ldx     #8.
        jsr     rght8   ;fetch value lo
        ldx     tempx   ;recover index
        sta     value,x ;low order of value
        inx
        pla
        sta     value,x ;hi order of value
        ldx     tempx   ;correct index
        pla
        sta     channel,x       ;channel
        pla
        sta     station,x       ;station
        inx
        inx             ;point to next entry
        stx     tempx   ;save for next time
        ldy     #0
        lda     base
        clc
        adc     #32.
        sta     base    ;increment base addr
        lda     base+1
        adc     #0
        sta     base+1
        lda     temp
        clc
        adc     #32.
        bne     rept
;
; all done, go back
;
pau     psh     save    ;recover return addr
        rts
;
; Error routine - sets error flag
;
err     lda     #080
        sta     error
        jmp     ok
;
```

```
; rght8 - to recover up to 8 bits.
; Order is most significant bit on right.
; On entry Y is the index to the work area
; and X is the number of bits to recover.
;
rght8   lda     #0      ;start fresh
topr    asl     a       ;shift to save latest bit
        pha
        lda     @base,y ;get next bit
        bpl     zeror   ;decide what it is
        pla
        ora     #1.     ;stick in a 1
        pha
zeror   pla
        dey             ;point to next
        dex             ;count down
        bne     topr    ;count till done
        rts             ;return with result in acc
        .end
```

```
        .include poppsh.text
;
; via address equates
;
irb     .equ    0c200   ;port b i/o
ira     .equ    irb+0f  ;port a i/o
ddrb    .equ    irb+2   ;port b data direction
ddra    .equ    irb+3   ;port a data direction
t1l     .equ    irb+4   ;timer 1 low
t1h     .equ    irb+5   ;timer 1 hi
acr     .equ    irb+0b  ;aux control register
ifr     .equ    irb+0d  ;interrupt flag register
;
clear   .equ    048     ;bit pattern for clear code (#)
save    .equ    0       ;return addr save area
freq    .equ    2
id1     .equ    freq+1
id2     .equ    id1+1
id3     .equ    id2+1
;
        .proc   poll,4
;
; procedure poll(freq, id1, id2, id3: integer);
;
; Purpose of this routine is to generate polling codes and to
; control transmitter and receiver power.
;
; Sequence of events:
;
; 1. Parent calls
; 2. Turn on appropriate transmitter FREQ.
; 3. Wait 1.5 seconds.
; 4. Send a clear tone (#) for 40 ms, wait 40 ms.
; 5. Send ID1 for 40 ms, wait 40 ms.
; 6. Send ID2 for 40 ms, wait 40 ms.
; 7. Send ID3 for 40 ms, wait 40 ms.
; 8. Turn off transmitter.
; 9. Turn on receiver.
; 10. Exit.
;
; VIA usage:
; This routine expects there to be a VIA in slot 2.
; Port A bit 0 is for xmit F1.
;            1             F2.
;            2        receive.
;
; Port B bits 0-6 present the appropriate levels for
; the DTMF encoder chip.
;
        pop     save    ;save return address
        pla
        sta     id3
        pla
        pla
        sta     id2
        pla
        pla
        sta     id1
```

```
        pla
        pla
        sta     freq
        pla
        lda     freq    ;pickup xmit frequency
        sta     ira     ;turn on transmitter
        ldx     #0      ;set up for a delay
        jsr     delay   ;of 256 ms
        jsr     delay   ;and do it 6 times
        jsr     delay
        jsr     delay
        jsr     delay
        jsr     delay   ;for a total of about 1.5 seconds
        lda     #clear  ;send a clear tone
        jsr     send
        lda     id1     ;get and send first id digit
        jsr     send
        lda     id2     ;get and send second id
        jsr     send
        lda     id3     ;get and send third id
        jsr     send
        lda     #0
        sta     ira     ;turn off transmitter
        lda     #04
        sta     ira     ;turn on receiver
        psh     save    ;recover return addr
        rts
;
; delay loop - this loop provides a delay of n ms, where n in
; passed in X. Returns with X = 0.
;
delay   lda     #0e8    ;hex 3e8 = dec 1000.
        sta     t1l
        lda     #03
        sta     t1h     ;start timer
        lda     #040
del1    bit     ifr
        beq     del1    ;wait for countdown
        dex             ;countdown multiples
        bne     delay   ;loop back till done
        rts
;
; Send - sends out DTMF code of whatever is in the accumulator.
; Tone is held for 40 ms, followed by a delay of 40 ms.
;
send    sta     irb     ;start DTMF tone
        ldx     #40.    ;hold
        jsr     delay
        lda     #0      ;turn off code
        sta     irb
        ldx     #40.
        jsr     delay
        rts
;
        .proc   tinit
;
; procedure tinit;
; This procedure initializes the VIA in slot 2 for
```

```
; controlling transmitter and receiver functions.
;
        pop     save
        lda     #07f    ;7 bits on port B out
        sta     ddrb
        lda     #07     ;bits 0-2 on port A out
        sta     ddra
        psh     save
        rts
        .end
```

```
        .include poppsh.text
        .proc readclock
        .public date
;
; procedure readclock;
;
; Procedure to read the CCS 7424 clock in slot 4.
; Date and time are returned in the integer array
; date which must be declared in the global section
; of the calling program and have at least 15
; elements.
;
return  .equ 0  ;temporary storage for return address
        pop return      ;save return address
        ldy #030        ;index to clock registers
        ldx #0          ;index to data array
rept    sty 0c0c1       ;specify what we want
        lda 0c0c0
        lda 0c0c0       ;read three times to insure
        lda 0c0c0       ;valid data
        and #0f         ;turn off hi bits
        sta date,x      ;save in caller's array
        inx
        lda #0
        sta date,x      ;hi byte all zeroes
        inx
        iny
        cpy #03d
        bne rept        ;loop till done
        lda #02f        ;release clock hold
        sta 0c0c1
;
; mask off unneeded stuff from clock
;
        ldx #0a         ;only want two lowest bits
        lda date,x      ;of hours ten
        and #03
        sta date,x
        ldx #010        ;and days ten
        lda date,x
        and #03
        sta date,x
goback  psh return      ;recover return address
        rts
        .proc holdclock
;
; procedure holdclock;
;
; Procedure to put a hold on the clock so it can be read.
;
return  .equ 0
        pop return      ;save return address
        lda #030
        sta 0c0c1       ;place hold
        psh return      ;recover return address
        rts             ;return to caller
        .end
```

```
        .include poppsh.text
        .proc   receive,2
; Procedure receive(delay, intrv: integer);
;
; Multi channel receiver program for the Apple Pascal system.
;
; This program uses the 6522 VIA to control timing of input
; sampling and to receive input stream. Input is on VIA PB7,
; and found at J2-8.
;
; Adapted from Apple II Lisa version
;
; July, 1983
;
; Tom English   HVO
;
;
; Page Zero
;
save    .equ    0       ;return addr save
delay   .equ    014     ;initial wait of about 1.5 bits
intrv   .equ    016     ;time for one bit
wrk     .equ    018     ;various counters and pointers
wrk1    .equ    019
ind     .equ    01a
cnt     .equ    01b
times   .equ    01c
frst    .equ    01f
;
; VIA addresses
;
irb     .equ    0c200   ;data i/o register
ddrb    .equ    irb+2   ;data direction register
t1l     .equ    irb+4   ;timer 1 low
t1h     .equ    irb+5   ;timer 1 hi
acr     .equ    irb+0b  ;aux control register
ifr     .equ    irb+0d  ;interrupt flag register
ira     .equ    irb+0f  ;rec/xmit control
work    .equ    01F00   ;receive buffer
;
        pop     save    ;save return address
        pla
        sta     intrv+1
        pla
        sta     intrv
        pla
        sta     delay+1
        pla
        sta     delay
        lda     #45.
        sta     frst    ;time out after 2.25 sec
        ldx     #0
        txa
fill    sta     work,x  ;fill work area with zeroes
        dex
        bne     fill
        sta     acr     ;zero out timer
        sta     t1l
```

```
        sta     t1h
        sta     ind     ;and index
        lda     #08
        sta     times
go      lda     #0
        sta     irb     ;be sure everybody is down
        sta     wrk     ;set counters
        lda     #14.
        sta     wrk1    ;for a 50ms time limit
wait1   lda     irb     ;looking for a hi bit
        bmi     go1     ;here it is
        dec     wrk     ;but only wait 50ms for it
        bne     wait1   ;this loop takes 3.584ms
        dec     wrk1    ;one byte is not enough
        bne     wait1   ;so the outer loop goes 14 times
        jmp     return  ;then gives up
go1     ldx     delay+1 ;pick up timer lo
        ldy     delay   ;and hi
        jsr     shot    ;start timer
wait2   lda     irb     ;test input state
        bpl     go      ;went low too soon, start over
        lda     #040
        bit     ifr     ;test timer
        beq     wait2   ;loop till time out
;
; By now we have seen about 1.5 hi bits,
; so assume there is something coming in.
;
        lda     #31.
        sta     cnt     ;to count input bits
        lda     #0ff    ;force first bit hi
        ldx     ind     ;pick up work index
        sta     work,x
        inx
        txa
        pha             ;save for now
next    ldx     intrv+1 ;interval low
        ldy     intrv   ;and hi
        jsr     shot    ;restart timer
        pla
        tax             ;recover index
        lda     irb     ;get input state
        sta     work,x  ;save it
        inx
        dec     cnt     ;count down bits
        beq     done    ;enough for this group
        txa
        pha
        lda     #040    ;otherwise wait for next bit
wait3   bit ifr
        beq     wait3
        jmp     next    ;then get next bit
done    lda     #0ff
        sta     frst    ;so we remember something came in
        stx     ind     ;stash index
        dec     times
        beq     pau     ;escape after 8 channels
        jmp     go
```

```
return  lda     frst    ;if nothing came in yet
        bmi     pau     ;something came in, so get out
        dec     frst    ;otherwise count down
        bne     go      ;and continue looking for awhile
pau     lda     #0
        sta     ira     ;turn off receiver
        psh     save    ;restore return addr
        rts             ;return to caller
;
; shot - subroutine to start the timer in one shot mode.
; Timer count is lo in X, hi in Y.
;
shot    stx     t1l
        sty     t1h     ;this starts it
        rts
        .end
```

```
        .include poppsh.text
        .proc   send3,2
;
; procedure send3 (ascline: string, switch: integer);
;
; Purpose of this routine is to send a string out
; via the CCS 7710A asynchronous communucations
; card in slot 3.
; The switch argument is used to specify the protocol
; to be used for the operations.
; Switch = 0 means no protocol.
; Switch = 1 means EOB/ACK protocol.
;          In this case the routine will not return
;          to the caller until it receives an ACK
;          (hex 06).
; Switch = 2 means that we're done, so send a ctrl-z.
;
; Tom English - August, 1985  - HVO
;
; CCS 7710A equates
;
cmd     .equ    0c0b0   ;command register
status  .equ    0c0b0   ;status register
data    .equ    0c0b1   ;data register
;
ack     .equ    06      ;ACK character
save    .equ    0       ;save area
staddr  .equ    02      ;string address
prot    .equ    04      ;protocol switch
retrn   .equ    0d      ;CR character
ctrlz   .equ    01a     ;ctrl-z
;
        pop     save
        lda     fsw     ;check first time switch
        bne     go      ;branch around initialization code
        lda     #023    ;acia reset
        sta     cmd
        lda     #011    ;characteristics
        sta     cmd
        sta     fsw
go      pla
        sta     prot    ;save the protocol switch
        pla
        pla
        sta     staddr  ;save string address
        pla
        sta     staddr+1
        lda     prot    ;first check for done.
        cmp     #2
        bne     reg
        lda     #ctrlz  ;ctrl-z
        jsr     wait    ;send it
        jmp     done    ;and get out
reg     ldy     #0
        lda     @staddr,y ;get string length
        tax
        beq     eos     ;null string breaker
        iny
```

```
loop    lda     @staddr,y ;pick up a character
        and     #7f       ;turn off sign bit
        jsr     wait
        iny
        dex
        bne     loop      ;continue till done
eos     lda     #retrn    ;set up to send a CR
        jsr     wait
        lda     prot      ;check protocol
        beq     done      ;get out if none
pwait   lda     status
        and     #1        ;check for input
        beq     pwait     ;wait till we get something
        lda     data      ;look at it
        cmp     #ack      ;is it an ACK?
        bne     pwait     ;no, keep waiting
done    psh     save
        rts
fsw     .byte   0
;
; routine to wait for a clear spot then send the character
;
wait    pha
wait1   lda     status    ;look at status
        and     #3        ;isolate tx and rx bits
        beq     wait1     ;wait if not ready
        and     #1        ;check for input
        beq     blast     ;none, so go for it
        lda     data      ;otherwise get it out of the way
        jmp     wait1     ;then try again
blast   pla               ;coast is clear, so go
        sta     data
        rts
        .end
```

```
        .include poppsh.text
        .proc setclock
        .public date
;
; procedure setclock;
;
; Procedure to set the CCS 7424 clock in slot 4.
; Date and time are passed in integer array date
; which must be declared in the global section
; of the calling program.  Array contents and
; corresponding clock registers are as follows:
;
;               Array   Clock
; Item          Index   Register
; Year 10       0       3c
; Year 1        1       3b
; Month 10      2       3a
; Month 1       3       39
; Date 10       4       38
; Date 1        5       37
; unused        6       36
; Hour 10 + 8   7       35
; Hour 1        8       34
; Minute 10     9       33
; Minute 1      10      32
;
; To finish off the setting we have to stuff
; zeroes in register 31, and in the command
; register.
;
return  .equ 0  ;temp storage for return address
        pop return      ;save return address
        ldy #3c         ;index to clock registers
        ldx #0          ;index to date array
rept    sty 0c0c1       ;set clock register
        lda date,x      ;get data
        sta 0c0c0       ;stuff it
        sta 0c0c0       ;three times
        sta 0c0c0       ;and maybe it will stick
        inx             ;next piece
        inx
        dey
        cpy #31
        bne rept
        sty 0c0c1
        lda #0
        sta 0c0c0
        sta 0c0c0
        sta 0c0c0
        sta 0c0c1
        psh return
        rts
        .end
```

```
.macro pop
pla
sta %1
pla
sta %1+1
.endm

.macro psh
lda %1+1
pha
lda %1
pha
.endm
```