U.S. DEPARTMENT OF THE INTERIOR
U.S. GEOLOGICAL SURVEY

USER DOCUMENT

PAGEIT USER DOCUMENT
VERSION 1.1
9/12/92

BY

ALEX BITTENBINDER

OPEN-FILE REPORT 92-538

Menlo Park, California
1992

PAGEIT USER DOCUMENT
Version 1.1
9/12/92

User Document

1. FUNCTIONAL DESCRIPTION

The system is capable of performing the following functions:

1.1. GENERAL

PAGEIT acts as a link between various computers and an alphanumeric paging service. Messages sent to the system via RS232 lines will be forwarded to specified pagers.

In addition, the system performs monitoring service for various computers by listening to specified 'heartbeats', and issuing appropriate pager messages if the 'heartbeat' ceases.

Inputs: Accepts ASCII pager messages up to 200 characters each, and state-of-health 'heartbeats' on up to eight RS232 lines. Supplies a user interface to maintain pager lists.

Outputs: Sends ASCII pager messages to various specified paging services via dial-up modem link. These messages are then transmitted by the paging service, and can be received by suitable alphanumeric pagers.

1.3. PAGER LISTS

PAGEIT maintains lists of pager id numbers linked to a given service, and the names of people carrying such pagers. One such list is maintained for each paging service used. A user interface Permits maintenance of these lists.

1.4. PAGER GROUPS

The software permits pagers to be associated into groups with a specified group name. Pager messages sent to the system can include a group name specifier, which will cause the message to be sent to all pagers on that group.

A given pager can belong to any number of groups.

A group may contain pagers belonging to various paging services.

1.5. STATE-OF-HEALTH MONITORING

Any of the computers connected to PAGEIT can be configured as 'patients'. 'Patient' systems are expected to issue a periodic 'heartbeat' message over the RS232 link. PAGEIT will monitor the 'heartbeats' for each of its 'patients', and issue an appropriate warning pager message if the heartbeat is not received within the specified time inerval.

On startup, PAGEIT reads a configuration file of 'patient' system names and associated heartbeat intervals (in seconds). A timer is maintained for each 'patient'. When a 'heartbeat' message of the form

"alive: <patient>"

is received, the timer for that 'patient' is reset. If the timer runs out before the next heartbeat is received, a message of the form:

"<patient> not responding"

is sent to the pagers belonging to the group "operations", and its timer is reset.

1.6. GROUP PAGING

A message may optionally start with a string of the form:

"group: <group name> ..."

If so, the message will be sent to all pagers belonging to the group <group name>. If the group name is not recognized, the message will be sent to all pagers belonging to the group "default".

If no group is specified, the message will be sent to all pagers belonging to the group "default".

If an individuals' name is specified instead of a group name, and an entry for that name exists, the message will be sent to that individual's pager only.

2. SYSTEM OVERVIEW

The initial configuration at USGS Menlo Park consists of two identical PC-based systems, as described below. One system is connected to 'andreas', the other to 'thebeach'. There is no cross-linking between the PC's; that is, the PC's are not aware of each other's operations.

It is anticipated that a later version will consist of a primary and a backup PC, with end-to-end confirmation by having the secondary PC confirm the page via an integrated paging receiver.

2.1 HARDWARE

The basic hardware consists of an AT-class compatible computer. The system includes a 3 1/2" and 5 1/4" floppy, and an internal hard disk. The system includes two add-in boards:

An internal modem (Hayes compatible). This is used to establish a phone link with the paging service computer.

An 8-port RS232 board (Digiboard PC/8). This provides up to eight input ports used for monitoring various systems. It is configured to present it's first port as COM3, etc. The connectors coming from the board are labelled P1 through P8, but are seen by the software as COM3 through COM10.

NOTE: As a result, the RS232 connector labelled P1 corresponds to port 3, P2 to port 4, etc.

2.2 SOFTWARE

The software consists of four major components:

2.2.1 OPERATING SYSTEM

The operating system is standard MS-DOS v5. Several useful utilities are available in addition to the standard DOS commands:

* DOSSHELL
  A crude window (character based window) environment for manipulating files and basic dos commands. <Shift><Tab> changes the active window, as shown by the color of the window title. Directories, and files within directories can be selected within the active window via the up and down arrow keys. A useful feature of this program is that <F9> causes the contents of the selected file to be displayed on the screen.

* EDIT
  This is a simple screen editor. It is invoked by the command

  ```
  EDIT <filename>
  ```

  Where 'filename' is the name of the file to be edited.

2.2.2 SAMPAGE

This is a vendor-supplied package from Teknow (800 279-9700) from Phoenix, Arizona. It resides in the subdirectory SAMPAGE, and consists of three components:

* A user interface which uses interactive screens to maintain various pager lists, which are stored in files in C:\SAMPAGE.

* An application program interface (API), in the form of a C-callable subroutine called "sendmsg". This routine provides the linkage between the custom C code PAGEIT, and SAMPAGE. Amongst other things, it creates message files containing outgoing paging requests.

* A TSR which searches for message files created by the API, places a phone call via the modem, and performs the required protocol to the paging service.

2.2.3 CUSTOM CODE

This is a program called PAGEIT stored in the directory C:\PAGER, under the file name PAGEIT.EXE. The program uses a configuration file named PAGEIT.CNF to read RS232 port parameters and the names of systems for which state-of-health monitoring is to be performed. This was written by Alex Bittenbinder, using Microsoft C v5.1. It listens to the RS232 ports for messages, performs the 'keep-alive' functions, and issues calls to the SAMPAGE API.
It also maintains a log file of exception reports on drive D:.

2.2.4 RS232 MULTI-PORT DRIVER

The 8-port Digiboard is controlled by a software driver from Greenleaf Software Inc. (214) 248 2561, in Dallas, Texas. This driver permits simultaneous acquisition of RS232 characters while the software is performing other functions.

2.2.5. HARD DISK DIRECTORIES

The hard disk contains two partitions, C: and D:. The C: root directory and three subdirectories:

* C:\DOS
  This contains the files for the DOS operating system. This is stock version 5.0 from Microsoft.

* C:\PAGER
  This contains the program PAGEIT.

* C:\SAMPAGE
  This contains the vendor supplied package SAMAPGE, including the user interface and the TSR which performs the paging protocol.

The D: partition contains only one file, the log file generated by the PAGEIT custom code. Note that this file grows with time, and has to be managed.

3. ROUTINE OPERATIONS

3.1. IS IT ALIVE

The system is configured to come to come up running when the system is rebooted, either via keyboard or by applying power. When PAGEIT is running, it displays a dynamic line of text at the bottom of the screen. If that line is static, the system is down.

If the line is active, it does not guarantee that all is well. There are recoverable errors which are logged into various error log files, but which permit the system to keep running. In addition, note that certain error conditions are not detectable with the existing system, such as lack of dial tone.

If PAGEIT has crashed, save the log file (D:PAGEIT.LOG) onto a separate floppy for crash analysis. Also look in the directory C:\SAMPAGE for files of the form *.ERR. Save any such files which may be relevant.

Attempt to restart the system by turning the power off, waiting five to ten seconds, and turning power back on. If the system does not come up, or issues error messages on startup, there has probably been a hardware failure.

3.2. ISSUING MANUAL PAGING REQUESTS

Page requests are normally received from the computer systems connected to PAGEIT. However, page messages can also be generated from the PC keyboard. To do this, press <alt><P> on the PC keyboard. This causes a window to pop up which presents a message form which can be filled in. The 'For:' and the 'Message:' fields are required; all others are optional. Up and down arrow keys move from field to field. When the 'For:' field is selected, the PgUp and PgDn keys scroll through the various pager names known to the system. Pressing F10 sends the message. Pressing Esc aborts the process.

3.3. MAINTAINING PAGER LISTS

The software package SAMPAGE includes an interface which permits maintenance of pagers lists for various paging services and lists of pager groups. It can be invoked by stopping the paging program (by typing 'quit' on the keyboard), and then entering the command

```
SAMPAGE
```

Its' operation is fairly obvious from the display. Selections are made via the <down arror>,<up arrow>, <pg up>,and <pg dn> keys. Note that at some points, <Enter> has to be pressed to activate a selected feature.

Be sure to perform the 'Save' function after changing the configuration in any of the categories below. There is no global 'save' at the end of the session. To restart the system after using the SAMPAGE interface, simply reboot the PC.

The following menu items are available:

Terminals: This lists the various Pager Service Companies ("Terminals") in use, the pager numbers leased from each company, and the names of the individuals carrying those pagers. The window displays one Terminal at a time.

Users: This lists the persons names and associated pager numbers for each Terminal. Various Terminals can be selected with the PgUp and PgDn keys. After selecting a terminal, pressing "Enter" will display the pagers in use with that Terminal. Entries can be changed by over-striking existing entries. New entries can be added at the bottom. Entries can be removed by over-striking with spaces.

Groups: This display works similarly to the Users display, except that various pager groups and their members are shown.

The group scheme works as follows: Some computer connected to the paging system will issue a paging request. The start of that message may include the name of the group to which this message is to be sent. When the PC receives the message, it will search for a matching group name, as defined by the SAMPAGE interface. If a matching name is found, the message is sent to all pagers belonging to that group, as specifed via the SAMPAGE interface. If no matching group name is found, an error is logged, and the message is sent to the group named 'default'. The SAMPAGE interface must therefore have groups defined which match the group names which will be requested by the originating computers.

PAGEIT must be configured with two groups with predefined meaning:

'default' is the name of the group to which messages will be sent if the message did not specify a group name, or if the group name was not created through the SAMPAGE interface. This group must be defined in SAMPAGE, and should contain whatever pagers should receive such messages.

'operations' is the name of the group to which 'obituaries' will be sent. These messages are generated within the PC, and are the result of some system failing to produce its heartbeat. A group by this name must be defined via the SAMPAGE interface. It is intended that this be the group of individuals concerned with keeping things running.

See the SAMpage User Manual for further information.
After making any changes, create new backup floppies as shown below.

3.4 EDITING THE CONFIGURATION FILE

PAGEIT reads certain parameters from a configuration file named PAGEIT.CNF in C:\PAGER. This file specifies RS232 input port parameters, the system names for which 'keep-alive' services are to be performed, and their heart beat intervals. An example of the file is shown below:

```
ports: 3
3 9600 N 8 1
4 9600 N 8 1
5 9600 N 8 1
patients: 4
andreas 180
prieta 60
killroy 10
thumper 45
```

The parameters for each port are:

COM port number:	COM1 and COM2 are used by the internal modem. The first available port is thus COM3, specified as '3'. Note that this corresponds to the first port of the Digiboard, labelled as P1 on the connector coming from the PC..
Baud rate.	
Parity:	where E is even parity, O is odd parity, and N is no parity checking. Other, exotic parity modes are supported. See the Greenleaf CommLib documentation.
Data bits.	
Stop bits.	

Following the port specifications are the 'patient' system names and the associated heart beat periods in seconds. The format of the file is crude in that the number of patients stated on the first line has to match the number of patient lines following.

The configuration file is a standard DOS ASCII file. The simplest way of editing the file is with EDIT. To do this, make sure you are in the directory C:\PAGER, and type the command

```
EDIT PAGEIT.CNF
```

Help is available by pressing <Alt><H>.

Create a new backup after making any changes, as outlined below.

3.5 CREATING A BACKUP

After making any changes to any files, create new backup floppies as follows:
Go the root directory c:\. This can be done by entering:

```
CD C:\
```

Insert a blank formatted floppy (either 5 1/4" or 3 1/2") and execute the command:

```
BACKUP C:*.* x: /S
```

Where x = A for 5 1/4" floppies, and
x = B for 3 1/2" floppies.
Insert additional floppies as requested.

3.6. MAINTAINING THE LOG FILES

Two types of log files are maintained:

PAGEIT maintains a log of exception reports. As configured, this file is found as:

```
D:\PAGEIT.LOG
```

This file can be examined with EDIT, copied to a floppy disk, or deleted. The file grows with time, and has to be periodically removed.

Note that the location of this file is specified by the second argument in the command which invokes PAGEIT. If nothing is specified for the second argument, the log messages will be produced on the screen.

4. ERROR REPORTING

A paging request passes through two layers of software: PAGEIT and SAMPAGE. Each is capable of detecting and logging errors.

PAGEIT logs errors in the file

```
               D:\pageit.log
```

This file can be examined with the editor. Stop the system by typing 'quit', and inoke the editor by typing

```
               edit d:\pageit.log
```

The file can be examined by scrolling with the up and down arrow keys, or via the PgUp PgDn keys.

SAMPAGE reports errors by displaying an error window on the screen for 5 seconds. This is too short to study the text, but is enough to indicate that an error has occurred. More significantly, it writes error information in files with names of the form

```
                    MM-DD-YY.ERR
```

All errors for a single day are stored in the appropriate file. The files are stored in the C:\SAMPAGE directory, and can be examined via the editor as above, or simply typed to the screen with the 'type' command.

In addition, the SAMPAGE interface maintains a log of the last 400 pager messages to be processed. This can be accessed via the SAMPAGE interface with the View token.

5. CRASH RECOVERY

If the system software is destroyed or corrupted (e.g. a hard disk failure, or operator error), the following procedure can be used to restore the system software:

5.1 RESTORE DISK PARTITIONS

If the hard disk was completely lost, establish two partitions, c: and d:. This procedure requires experience with loading DOS.

5.2 LOAD DOS

This can be done by booting from the PAGEIT BOOT floppy. Insert this floppy into drive A and boot the system, either by turning on power, or by pressing <Cntrl><Alt><Del>. After DOS comes up, perform the command

```
               FORMAT C:/S
```

This will remove any data from the C: disk, format it, and move the bootable portion of the operating system to it.

5.2 LOAD THE SYSTEM SOFTWARE

Go to the root directory ('cd c:\'). Insert the first PAGEIT BACKUP floppy, and restore all remaining files and subdirectories by entering

```
RESTORE x: C: /S
```

Where x = A for 5 1/4" floppies, and
      x = B for 3 1/2" floppies.

Insert additional backup floppies as requested.

5.3 REBOOT

Press <cntrl><alt><del> to reboot the system. It should come up and run. If not, either the backup floppies are corrupted, or there is a hardware problem.

PAGEIT
PROGRAMMERS NOTES
VERSION 1.1
9/12/92

TABLE OF CONTENTS

Programmer Notes

1.0 HISTORY

This program was requested to act as a replacement for the SLC II watchdog systems. There is urgency to this request, as the SLC's are no longer maintainable. The objective is to produce a replacement for the SLC's quickly. It appears that there are things on the horizon (untethered computing) which will make this thing obsolete fairly quickly. It is hoped that this effort can be upgraded to meet future requirements.

A search of commercially available software has been made. It appears that a number of companies are currently working on systems which will meet our requirements. None are ready at this time, but several will be soon. In particular, Fitzgerald Telecom is working on a product which would meet our requirements almost completely. Unfortunately, they are in the beta test stage, and it appears that we may not participate due to federal restrictions. In addition, Motorola is developing several products and services which will be applicable.

2.0 SOFTWARE COMPONENTS

The system runs under DOS. This was hard to avoid given the time pressure. No paging protocol (IXO) software running under any other system could be found, and given the time constraint, in-house coding of this function seemed prohibitive. DOS v3.1 was used as a development platform. The target system is based on MsDOS v5. Microsoft C v5.1 was used. In addition, several commercial software packages were used:

2.1 PAGING SOFTWARE

SAMpage, version 2.69, and the associated API. Available from Teknow, (800) 279 9700, Phoenix, Arizona. This consists of three components:

1 User Interface: Found in C:\SAMPGE\PGSETUP.EXE. This is a DOS based application which maintains lists of pager numbers, names, and paging services. Includes COW (character oriented windows) displays permitting maintenance of pager lists by fairly intuitive means. It produces a series of files.

2. TSR: This remains resident, and continually looks for message files in c:\SAMPAGE\ with a .MSG extension. When one is found, it dials the specified paging service, and sends the message. The IXO protocol for alphanumeric paging is used.

3. SENDMSG: This is the programming interface. It's a C-callable routine which accepts a message, and generates files which are grabbed by the TSR.

Programmer Notes

Version 2.69 has been produced at our request. It includes several bug fixes in the group maintenance code, and it has Teknow's copy protection scheme removed. This permits the use of DOS backup and restore to be used on the entire software package.

In addition, it sports a new argument which controls how long the TSR sleeps between searches for files to page out (C:\SAMPAGE\*.MSG). In the stock version, this is something like 15 seconds, which was considered to be a bit long.

Another new feature has been added to PGCALL. If PGCALL is invoked with the command line

```
          c:\sampage\pgcall /vn
```

it will display it's version number. The version in use echoes "v2.6".

Caution on version contol: The printed documentation and the software banner declare v2.6. However, (at Teknow's suggestion) I have declared this version to be v2.69, and so marked the floppies and manual. The serial number of the floppies used start with 269. The complete distribution consists of two floppies: SPN2691101 which is Sampage, and SPA2691106 which is the corresponding API. The older APIs (version 2.6) will not work with v2.69.

There is an annotated SAMPAGE manual, hopefully with this document, which is marked in handwriting as V2.69 on the cover. It contains handwritten notes on the new features. Basically, the new feature is a new argument on the command line to PGCALL. This is

```
                    T9=n
```

where n= number of seconds the TSR sleeps between searches for .MSG files. It is currently set to 1.

2.2 RS232 DRIVER

Greenleaf CommLib. Rev S3-3.20B.
Greenleaf Software, Dallas Texas
(214) 248 2561.

This is an interrupt driven RS232 driver with lots of bells and whistles. This is used to get around the fact that DOS does not buffer input from the RS232 ports while the processor is off executing applications code. This meant that incoming messages could be lost while we were processing a previous message.

Two benefits are gained by using this: we can stack up incoming messages while doing other things, and it provides support for add-on boards capable of servicing numerous RS232 ports, if that should be required in the future.

2.3 CUSTOM CODE

The program PAGEIT was written in-house under Microsoft C v5.1. It picks up messages from the RS232 port, and calls SENDMSG to issue the page. Errors are logged to a file specified in the second argument of its calling sequence (currently D:pageit.log). It also provides a visible 'I'm alive' line on the screen.

In addition, the program will monitor heartbeats for a list of 'patients'. If a patient's heartbeat is not received before a specified time interval, an 'obituary' page message of the form

<sys name> not responding

will be issued.

3.0 HARDWARE

Minimal hardware performance is required for this task. The target system is an Everex Step 286, small hard disk, both kinds of floppies, an internal modem, and a Digiboard PC/8, which offers eight RS232 ports.

A Everex 24 internal modem, as well as a generic Hays-compatible modem w used with no problems. The SAMpage configuration program supports a generous variety of other modems.

4.0 TARGET SYSTEM DIRECTORY STRUCTURE

The following directories and files are on the target system. The intent is to include all files required for re-compilation. In addition, the D: partition is used to store the PAGEIT log file.

4.1. ROOT

```
 Volume in drive C has no label
 Volume Serial Number is 1922-18FF
 Directory of C:\

COMMAND  COM     47845 04-09-91   5:00a
CONFIG   SYS        71 07-20-92  11:20p
AUTOEXEC BAT       406 07-28-92   6:11p
ROOT     DIR       425 04-17-92  11:11a
DOS          <DIR>     07-28-92   5:57p
SAMPAGE      <DIR>     07-28-92   6:00p
PAGER        <DIR>     07-28-92   6:00p
SAMPAGE  BAT        24 07-28-92   6:05p
        8 file(s)      48771 bytes
                    30339072 bytes free
```

4.2. DOS: This contains the DOS system files

4.3. C:\PAGER:

The in-house PAGEIT program. Note that there will also be *.DOC files, which do not show in the listing below. The *.DOC files are being written as this listing is obtained. Included are the source, make and link files. SET_PG.* is a routine to get the command line variables to PAGEIT. SPAIP*.* is the Teknow API stuff.

```
 Volume in drive C has no label
 Volume Serial Number is 1922-18FF
 Directory of C:\PAGER

.            <DIR>      07-28-92   6:00p
..           <DIR>      07-28-92   6:00p
PAGEIT   EXE     77576 07-28-92   4:19p
PAGEIT   CNF        94 07-28-92   2:58p
SET_PG   C        1288 05-21-92   1:19p
PAGEIT   C       14398 07-28-92   4:19p
PC8_SET  C       11100 05-26-92  11:55p
PAGEIT             232 05-22-92   1:14p
PAGEIT   LNK        78 07-28-92   2:29p
GFCS     LIB    191549 11-14-91   3:20a
SPAPI    H         432 10-24-90   2:21p
IBMKEYS  H       18652 11-14-91   3:20a
ASCIIDEF H        1627 11-14-91   3:20a
ASIPORTS H       42601 05-20-92   6:35p
VIDEO    H        4471 11-14-91   3:20a
       15 file(s)     364098 bytes
                    30339072 bytes free
```

4.4. C:\SAMPAGE

The Teknow software. These files will change as pagers are added. Note that the software maintains a log of messages sent. This log is circular, keeping the last several hundred messages set. It also maintains daily error logs, *.ERR. See the SAMpage documentation.

```
 Volume in drive C has no label
 Volume Serial Number is 1922-18FF
 Directory of C:\SAMPAGE

.            <DIR>      07-28-92   6:00p
..           <DIR>      07-28-92   6:00p
PGORIG   COM     13934 07-10-92   2:59p
PGCALL   EXE     58097 07-10-92   2:59p
PGSETUP  EXE     49812 07-10-92   2:59p
PGKEYS   EXE     14171 07-10-92   2:59p
PGLOAD   BAT        31 07-21-92  10:28a
TERMINAL DAT      1400 07-21-92  10:35a
CONTROL  DAT         3 07-29-92   2:52p
LOG      DAT    124807 07-29-92   2:52p
SIZE     DAT         5 07-21-92  10:28a
GROUPS   DAT      2800 07-28-92   3:07p
PAGEIT   EXE     77636 07-28-92   4:15p
USERS    DAT      2601 07-28-92   3:07p
       14 file(s)     345297 bytes
                    30339072 bytes free
```

5.0 PAGEIT NOTES

The program consists of some initialization code followed by two nested loops. The outer loop (working loop) is infinite. The only way out is if someone types the characters 'quit' on the keyboard. We then terminate with a big fuss to impress the human. The working loop consists of a message gathering loop, and message processing code. The working loop is traversed each time a complete message is received.

The message gathering loop is traversed periodically. At the top of this loop is a call to the Greenleaf RS232 driver. This call includes a variable specifying the number of milliseconds we will hang in the driver before returning. This can be used to control the rate at which we run through the loop. The driver will return to us any characters which came in since the last time we called the driver. We then append these characters to a message string. The driver will tell us if a message terminator (currently a #) has been received.

If so, we drop out of the message gathering loop, and traverse the rest of the working loop. This consists of the message processing code, which includes:

* See if the message includes the string 'group:' if it does, we search for a group name. The page will then be directed at that group. If we don't find anything, we set the group name to 'default'. Starting with version 1.1, we assume that the 'group:' keyword and group name are at the beginning of the message. Both are stripped off before the message is sent out.

* See if the message is a heartbeat. That is, does it contain the string 'alive:'. If so, we try to pick up the patient's name, and reset its 'last_heard' time. If the message was a heartbeat, we do a 'go to' to the bottom of the working loop. The goto is used to freak out the C snobs.

* Otherwise, we send the message as is via 'sendmsg', the API from Teknow. In the case where no group was specified, the group name is set to 'default'.

Note that the message gathering loop is the place to stick any calls to routines which have to be called regularly. e.g. there is talk of having this thing monitor contact closures and voltage levels.

The program accepts command line arguments, and reads a configuration file. The command line arguments are:

Directory:	Path to where the SAMpage files are located. The *.MSG files containing messages to be sent out will be placed here.
Log File:	Optional. If supplied, PAGIT will write status and error messages to that file. If not supplied, log messages will be sent to the screen. The system is configured with a log file argument of d:pager.log. Nothing else resides on the d: partition.

The parameter file specifies the ports we're to monitor, and the names and heartrates of our 'patient' systems. The format is crude. We read the count for ports and patients, and then expect there to be that many lines. For example:

```
ports: 3
3 9600 N 8 1
4 9600 N 8 1
5 9600 N 8 1
patients: 4
andreas  10
killroy 20
squat 30
humper 10
```

This says we're to monitor the first three ports of the Digiboard. We also have four patients, with the stated names, and heart beat periods in seconds.

Heart Monitoring:

We do this by setting an array called last_heard to the current time. Within the message gathering loop, we call the routine 'call_the_dead'. If the time in last_heard is older than the stated heart period in the configuration file, we issue an obituary message to the pagers. The other piece of the scheme is that if we get a message containing the string:

```
               alive:
```

we scan for a system name, try to match it with a known patient name, and if found, set it's last_heard time to current time.

Digiboard setup:

The Digiboard PC/8 is initialized by the routine setv_pc_digigoard. This is supplied by Greenleaf. The only change made to this routine is to use IRQ5 instead of IRQ3. The jumpers on the board are set accordingly. Note that the graphics showing jumper setting in the comments have not been changed.

NOTE: The first Digiboard port (which the board calls 1) is COM3, and is referred to as 3 throughout the software.

Group Paging:

The Sampage API accepts a 'group' argument. The Sampage interface permits the user to group pagers into named groups. Calling the API with a group name will cause the message to be sent to all pagers belonging to that group. The group name can also be an individuals name.

What we do is scan messages for the string 'group:', and grab the following word as the group name. There is a weakness here, in that we have no way of knowing whether that group name exists in the Sampage stuff. Someday we should get so we can read Sampage's setup files and check that the group name is legitimate.

If no 'group:' specifier is found, we set the group name to 'default'. It is presumed that the administrator has arranged there to be such a group.

6.0 CONTROL FILES

6.1 THE MAKE FILE: PAGEIT:

```
pageit.obj: pageit.c
     cl /c /Od /Zi /Fs pageit.c

set_pg.obj: set_pg.c
     cl /c /Od /Zi /Fs set_pg.c

pc8_set.obj: pc8_set.c
     cl /c /Od /Zi /Fs pc8_set.c

pageit.exe: pageit.obj  set_pg.obj pc8_set.obj
     link @pageit.lnk
```

6.2 LINER FILE (PAGEIT.LNK)

```
/STACK:3072 /CO pageit+set_pg+pc8_set+spapis5
pageit.exe
pageit.map
gfcs
```

Note: 'gfcs' is the Greenleaf RS232 port driver library.

'spapis5' is the Sampage API; the last s means small memory model, the 5 means Microsoft C v5.1

6.3 TARGET SYSTEM AUTOEXEC.BAT

```
C:\SAMPAGE\PgOrig
REM: /DISK_ERR          log errors to disk,
REM: /SCREEN 5          show error messages for 5 seconds,
REM: T9=1               TSR to check for chores once per second,
REM: /FAST              Dont initialize modem between calls
c:\sampage\PGCALL /DISK_ERR /SCREEN 5 /T9=1 /FAST
@ECHO OFF
PROMPT $p$g
PATH C:;C:\DOS
SET TEMP=C:\DOS
cd c:\pager
pageit c:\sampage d:pageit.log
```

6.4 TARGET SYSTEM CONFIG.SYS

```
DEVICE=C:\DOS\SETVER.EXE
DEVICE=C:\DOS\HIMEM.SYS
DOS=HIGH
FILES=10
```

Appendix A

```
// PAGEIT  - to get a message via RS232 and send it to an alpha pager
//
//Usage:
//  PAGEIT <Directory where Sampage is installed> [<log file>]
//
// This program will listen to RS232 ports from a Digiboard Digichannel
// PC/8 board,  and expect to receive  messages to be issued to
// alphanumeric pagers.

//Port configuration parameters are read from a configuration file "pageit.cnf"
//Who knows what they'll be.....
//
// It uses the Greenleaf COM port driver. This is an interrupt driven driver
// which can accept characters while the main program is off somewhere else.
// Note that DOS by itsself does not do that.
//
// It then calls a routine supplied by Teknow, called Sampage API, which
// generates a file for a TSR which executes the page.
//
// Any weird happenings will be written into the log file. Note that the
// log file is optional. If not stated on the command line, I'll write log
// messages to the screen.


#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "asiports.h"
#include "asciidef.h"
#include "ibmkeys.h"
#include "spapi.h"    // This is the pager calling stuff, from Teknow

                        // GENERAL GLOBAL STUFF
#define CONFIG_FILE "pageit.cnf"
#define MAXMSG 500  // max message length
char logmsg[200];// for assembing log messages

                        //  SYSTEM DEAD DETECTION STUFF
#define MAXSYS 10                 // Max number of system to watch over
#define MAX_NAM_LEN 20        // maximum name length of system to watch
int maxquiet[MAXSYS];         // heart rates (periods, actually) of our patients
char *alive_key="alive:";     // key on this for alive messages
int alv,igr,igrnm;            // return from string searcher
int n_patients; // Number of patients to watch over
char *patient[MAXSYS]=  // names of systems which we monitor
{
    "                    ",
    "                    ",
    "                     ",
    "                     ",
    "                     ",
    "                     ",
    "                     ",
    "                     ",
    "                     ",
    "                     "
);
char tmp[100];                  // for temp storage of stuff
char *dead_msg = " not responding";
time_t last_heard[MAXSYS];      // timers to see who's died

                         //  RS232 CONFIGURATION STUFF
#define RD_HANG 500             // hang in asynch read routine for yea many milli-
#define TERMINATOR '#'   // Character which terminates a message
typedef struct                  // our private little port structure
    {
    int port_number;
    long baud;
    char parity;
    int bits_per_char;
    int stop_bits;
    PORT *grnlf; // pointer to the Greenleaf port open structure
    int ret;  // return from the Read statement (for detecting msg complete)
    char string[MAXMSG];
    } PORT_PARAM;
PORT_PARAM in_port[MAX_PORT];
PORT_PARAM *iprt;
int n_ports;  // number of ports in use


                              // PAGER GROUPS STUFF
char *page_dir;               // where the Sampage software is installed
static char operations_grp[]=           "operations"; //to where we send obituar
static char default_grp[]=         "default"; //group when no group is specified
char group_key[10]=                          "group:"; // after this comes group

#define MAX_GRP_L 80
char group[MAX_GRP_L]; // name of user group, as defined in Sampage
char pgr_err[7][50] =
    {
    "Message too long",
    "Cannot open group file",
    "Cannot find user",
    "Could not assign new message number",
    "Could not write message",
    "Could not update SAMPAGE.DAT file",
    "Sampage not loaded"
    };
```

```
void main(int argc, char **argv);

void main( argc, argv)
int argc;
char **argv;
{
    static char message[MAXMSG+1];// the assembled message to be paged
    char *msg_body;// after various things have been peeled off the message
    static char newstr[MAXMSG+1];// what we slurp with each read

    int ret;
    int opnret[MAX_PORT]; // return values from reads
    int i;
    char kbd[4]; //buffer for keyboard messages (currently just 'quit')

    //get command line arguments
    if (set_pg(argc, argv, &page_dir) < 0)
         {
         printf("Bad arguments. Usage:\n");
         printf("PAGEIT <pager directory> [< log file >]\n");
         log("PAGEIT terminating; bad arguments; set_pg error");
         exit(-1);
         }

    // read configuration parameters
    if(read_config() <=0)
         {
         log("PAGEIT terminating; bad configuration file");
         exit(-1);
         }
// proudly display configuration paramters
// printf("Tending %d patients:\n",n_patients);
// for(i=0; i<n_patients;i++)(printf("patient %d: %s %d\n",i,patient[i],maxquie
//  printf("\nListening on %d ports. Paramters:\n",n_ports);
//  for(i=1;i<=n_ports;i++)
//       {
 /       iprt=&in_port[i-1];
//                  printf("port:%d, baud:%ld, parity:%c, data bits:%d, stop bit
//                                      iprt->port_number,
//                                      iprt->baud,
//                                      iprt->parity,
//                                      iprt->bits_per_char,
//                                      iprt->stop_bits);
 /       }

    // Initialize timers for our patients
    for( i=0; i<n_patients; i++) time( &last_heard[i] );

    // Stuff below is needed in some mysterious situations.
//   If nothing comes to the screen activate the two statements below.
//setbuf( stdout, NULL );
//setbuf( stdin, NULL );

// GREENLEAF DRIVER INITIALIZATION.
// This will get us interrrupt driven IO
// Call below does some really heavy exotica with the Greenleaf driver.
// Amongst many things, the first port number used by the board
// is defined there
if(setv_pc_digiboard(n_ports)<=0)
     {
     log("Terminating. Error from setv_pc_digiboard");
     exit(-1);
     }
for(i=0;i<n_ports;i++)
     {
     iprt=&in_port[i];
     iprt->grnlf= PortOpenGreenleaf(    (iprt->port_number)-1,
                                                            iprt->baud,
                                                            iprt->parity,
                                                            iprt->bits_per_
                                                            iprt->stop_bits
     if ( (iprt->grnlf)->status < ASSUCCESS )
          {
  sprintf(logmsg, "Cannot Open COM%d - PortOpen() Returned %d\n",
             iprt->port_number + 1, (iprt->grnlf)->status );
          log(logmsg);
          exit(-1);
          }
     //printf("opened port %d at %ld\n",iprt->port_number,iprt->baud);
     }

// Announce start of operations
printf("\n\n\n\n\n\n\n\n                PAGING SERVICE ENABLED\n\n\n\n");
log("PAGEIT starting execution.");
printf("Note: If bottom line is not active, system is down! \n");
printf("      Type 'quit' to terminate operations.\n");
printf("      Press <ctrl><alt><del> to restart\n\n\n");

// **************** THIS IS TOP OF THE WORKING LOOP. ********************
while(1)
     {
     // Loop below will gather characters until the terminating character
     // is received. Any characters coming in while we're off paging will
     // be buffered up. We hang for RD_HANG ms each time.
     message[0]=(char)0; // erase old message
     ret=ASBUFREMPTY;

     // //////////////// TOP OF MESSAGE GATHERING LOOP /////////////////////
     // update vital signs timers for our patients
// and issue obituaries for any stiffs.
while(1)
     {
     call_the_dead();

     // hang on each of the read ports for a bit, and come back with
     // whatever had dribbled in since the last time we checked
     for(i=0;i<n_ports;i++)
          {
          iprt=&in_port[i];
          iprt->ret= ReadStringTimed( iprt->grnlf, newstr,
                                                              MAXMSG, TERMI
                                             // TERMINATOR = character whic
                                             // RD_HANG = ms wait before ex
          (void)strcat(iprt->string,newstr);   //add it to what we alr
          //printf(" %d: %s\n",iprt->port_number,iprt->string);
          (void)alive();// show the world we're alive

          if( kbhit()!=0) // then someone pressed a key
               { // collect the last 4 characters, and see if it's "qu
               // shift the last three characters
               kbd[0]=kbd[1]; kbd[1]=kbd[2]; kbd[2]=kbd[3];
               kbd[3]=getch(); // new character
               kbd[4]='\0';
               if( strcmp(kbd,"quit")==0 ) // a quit from the keyboard
                    {
                    log("Terminated by keyboard request");
                    printf("\n\n\n         *** WARNING ***\n");
                    printf("YOU HAVE TERMINATED PAGING SERVICE!\nPress
                    (void)putch('\007');
                    (void)putch('\007');
                    (void)putch('\007');
                    exit(0);
                    }
               }
          if(iprt->ret==ASSUCCESS)
               {
               strcpy(message,iprt->string);// load message array
               strcpy(iprt->string,"\0");// clear the old stuff
               goto got_msg;// drop out of for loop
               }
          }
     }
     //////////////////// END OF MESSAGE GATHERING LOOP ///////////////

// -------------------TOP OF MESSAGE PROCESSING BLOCK --------------
// GOT A MESSAGE. WHAT TO DO WITH IT?
got_msg:
//printf( "\nMessage= %s\n", message );
//log(message);
```

```
// Fist, some housekeeping
msg_body=message; //assume we're shipping the whole thing
strcpy(group,default_grp);// set group name to default
// Is there a group id in this message?
igr=pat_match(message,group_key);
if( igr >=0 ) // then there is a group keyword in the message
     {
     igrnm=get_next(tmp,&message[igr+strlen(group_key)]);
     if(igrnm >=0 )
     // then there was a next word
          {
          igrnm++;igrnm++;// step to-,and over the group name terminat
          msg_body=&message[igr+strlen(group_key)+igrnm];
          strcpy( group,tmp ); // reset group id
          }
     else
          {
          log("Missing group name:");
          log(message);
          }
     }

// Is it an "Alive:" message?
alv=pat_match(message,alive_key);  // is there an "alive:" in this mes
if(alv>=0)  // its an I'm alive message - reset that systems' timeout
     {
     if( get_next(tmp, &message[alv+strlen(alive_key)]) >=0 )// got a
          {
          //printf("got alive message for %s\n",tmp);
          // is this anyone we know?
          for(i=0;patient[i]!=NULL;i++)
               {
               if( strcmp(tmp,patient[i])==0 )
                    {
                    //printf("\nReprieve for %s\n",tmp);
                    time( &last_heard[i] ); // reset it's timeout
                    goto did_this_msg ;
                    }
               }
          }
     else
          {
          log("No system name in alive message below:");
          log(message);
          }
     goto did_this_msg; // Yes - a taste of cold steel, gentelmen...
     }
// End of Alive processing

// And after all the violence and double talk ...
```

```
        //                          JUST DO THE PAGE
        {
             //  This is the API from Teknow. It will create a
             // file which will be picked up by their TSR
             // printf("\nSENDING MESSAGE %s TO GROUP %s\n",msg_body,group);
             ret = sendmsg(group,msg_body,page_dir) ;
             if (ret)
                  {
                  printf("\nError from sendmsg: %s; %d\n",pgr_err[-(ret+1)],re
                  sprintf(logmsg,"Error sending message: %s",pgr_err[-(ret+1)]
                  log(logmsg);
                  }
             }
        did_this_msg:;// come here after processing message
        // -----------------BOTTOM OF MESSAGE HANDLING LOOP ----------------
        }
    // ******************* BOTTOM OF WORKING LOOP ***************************
    }
// ++++++++++++++++++++++++++ END OF main() +++++++++++++++++++++++++++++++


// This routine sends out obituaries for any patients who've not reported
// alive messges, and resets the victim's timer. Thus obituaries will go
// out every maxquiet[i] interval.
call_the_dead()
{
    char ctmp[80];
    int i;
    int ret;
    time_t now;
    time(&now);
    for(i=0;i<n_patients;i++)
         {
         if( difftime( now,last_heard[i] ) > (double)maxquiet[i])
              {
              time(&last_heard[i]); // reset the dead systems timer
              //printf("sending death msg for %s\n",patient[i]);
              strcpy(ctmp,patient[i]); // assemble death notice
              strcat(ctmp,dead_msg);
              ret=sendmsg(operations_grp, ctmp, page_dir);
              if (ret)
                   {
                   sprintf(logmsg,"Error sending obituary: %s",pgr_err[-(ret+1)
                   log(logmsg);
                   }
              ctmp[0]='\0'; // wipe clean after using
              }
         }
}
```

```
//routine to tell the world we're alive
#define MAXSTR 100
static char banner[MAXSTR]={
//"Oh, Mama, can this really be the end"
"Pageit v1.1 Pageit v1.1 Pageit v1.1 Pageit v1.1 Pageit v1.1 Pageit v1.1 "
};
static int next=0; // pointer to next char to output

alive()
{
    int i;
    int maxstr;

    maxstr=strlen(banner);

    (void)putch(banner[next]);                            // put up left character
    i=next+1;
    while(i<(maxstr-next-1)){
         (void)putch(' ');                                //move to right
         i++;
         }
    (void)putch(banner[maxstr-next-1]);              // put up right character
    i=next+1;
    while(i<(maxstr-next) ){
         (void)putch('\b');
         i++;
         }
    next++;
    if( next==(maxstr/2+1) ) {
         next=0;              // start over
         (void)putch('\r');
         }
}



// routine to read in table of system to watch over

int read_config()
    {
    FILE *fh; // the file handle thingy
    PORT_PARAM *inprt;
    int n;
    char temp[80];
    int i,j ;
    fh=fopen(CONFIG_FILE,"r");
    if(fh==NUL)
         {
```

```
     printf("Cannot open %s; %s\n",CONFIG_FILE, sys_errlist[errno]);
     return(-1);
     }
// Read number of ports
i=fscanf(fh," ports: %d",&n_ports);     // number of ports in use
if(i<=0 || n_ports>MAX_PORT || n_ports<=0 )
     {
     printf("bad channel count. Returned %d\n",i);
     return(-1);
     }

// Read the parameters for each port
inprt=&in_port[0];
for(i=1;i<=n_ports;i++)
     {
     fscanf(fh," %d %ld %c %d %d",
                                   &(inprt->port_number),
                                   &(inprt->baud),
                                   &(inprt->parity),
                                   &(inprt->bits_per_char),
                                   &(inprt->stop_bits)       );
     inprt++;
     }
     inprt=NULL;

// Now look for "patients:", and read the number of patients to tend.
i=fscanf(fh," patients: %d",&n_patients);
if(i<=0)
     {
     printf("Bad config file; no patients, ret=%d\n",i);
     return(-1);
     }

// read patient names, driven by count from above
for (i=0;i<n_patients;i++)
     {
     j=fscanf( fh," %s %d",patient[i],&maxquiet[i] );
     if(j<=0)
          {
          log("error reading patient names");
          return(-1);
          }
     }
     patient[i]=NULL;
if(ferror(fh))
     {
     printf("Error reading system names; %s\n",sys_errlist[errno]);
     return(-1);
     }
```

```
    //All done with configuration file
    fclose(fh);
    return(1);
    }

// Return the first occurrence of str2 in str1. Return -1 if ot match found
// modified from Peter Darnell
int pat_match(str1,str2)
char *str1, *str2;
    {
    char *p, *q, *substr;

    for (substr= str1; *substr; substr++)
         {
         p=substr;
         q=str2;
         while(*q)
              if(*q++ != *p++)
                   goto no_match;
              return substr-str1;
         no_match:;
         }
    return -1;
    }


/////// routine to extract next word from str2 into str1
//a word is terminated by anything below "0", TERMINATOR, or ';'
// leading blanks will be ignored.
int get_next(to,from)
char *to, *from;
    {
    char *f, *t;
    int cnt;
    long i,j,k;
    cnt=-1;
    f=from;
    t=to;
    while (*f==' ')// skip leading blanks
         {
         f++;
         cnt++;
         }
    while(    (*f!=TERMINATOR)  &&       ( ((int)(*f)) >47 )  && (*f!=';') )
         {
         *t++=*f++;
         cnt++;
         }
    *t='\0';
```

```
return cnt;
}
```

Appendix B

```
//This routine was taken from the program Multloop, supplied by Greenleaf
//Software, (214) 248 2561, Texas. It is their stated method of documenting
//the setup of various boards which they support.

//void setv_pc_digiboard( int channel_count );
// The calling program, Multloop, had the following include files:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "ibmkeys.h"
#include "asiports.h"
#include "asciidef.h"
#include "video.h"

/*
 * This setv_xxxx routine is used to set up the Digiboard products for the
 * standard PC bus.  This actually means one of several boards which all
 * look more or less the same to the software.  In any case, it can include
 * up to 32 ports of Digiboard.  The port addressing scheme used by default
 * is set up to accomodate two Digiboard PC/16 boards in the same system.
 * The user cannot change any of the port addresses on these boards, so that
 * kind of makes them a sensible default.  What this means is that if you are
 * trying to set up a Digiboard COM/4 or COM/8, you probably want to use
 * the port mapping shown below:
 *
 *      Interrupt:                 IRQ3
 *      First Uart Address:        0x100
 *      Status Register Address:   0x140
 *
 * Note that the UART addressing scheme is a little complicated.  The setv_xxxx
 * routine below forces you into it though, unless you feel like modifying the
 * code.  The port addressing is forced into the scheme used by the /16 boards:
 *
 *    Board 0:    Port 1 :  0x100      Board 1:     Port 1 :   0x188
 *                Port 2 :  0x108                   Port 2 :   0x190
 *                Port 3 :  0x110                   Port 3 :   0x198
 *                Port 4 :  0x118                   Port 4 :   0x208
 *                Port 5 :  0x120                   Port 5 :   0x210
 *                Port 6 :  0x128                   Port 6 :   0x218
 *                Port 7 :  0x130                   Port 7 :   0x220
 *                Port 8 :  0x138                   Port 8 :   0x228
 *                Port 9 :  0x148                   Port 9 :   0x230
 *                Port 10 : 0x150                   Port 10 :  0x238
 *                Port 11 : 0x158                   Port 11 :  0x240
 *                Port 12 : 0x160                   Port 12 :  0x248
 *                Port 13 : 0x168                   Port 13 :  0x250
 *                Port 14 : 0x170                   Port 14 :  0x258
 *                Port 15 : 0x178                   Port 15 :  0x260
 *                Port 16 : 0x180                   Port 16 :  0x268
```

```
*
* What the addressing scheme means is that you can test two PC/16 boards
* using the default setup.  If you have a COM/4 or a COM/8 you need to
* set the DIP switches up to use these addresses.
*
* On the command line, you need to give a number of ports to test.  If
* you are testing a COM/4, type "MULTLOOP D 4", for example.
*
* Dip Switch and Jumper settings for a COM/8 are shown below:
*
*
*                                       -------------------------
*                                       | o     o     o  o  o  o | ON
*  Status port selection DS1: 0x140     |    o     o             | OFF
*                                       | 1  2  3  4  5  6  7  8 |
*                                       -------------------------
*
*
*                                       -------------------------
*                                       | o     o  o  o  o  o  o | ON
*  UART Port 1 selection DS2: 0x100     |    o                   | OFF
*                                       | 1  2  3  4  5  6  7  8 |
*                                       -------------------------
*
*
*                                       -------------------------
*                                       | o     o  o  o  o     o | ON
*  UART Port 2 selection DS3: 0x108     |    o              o    | OFF
*                                       | 1  2  3  4  5  6  7  8 |
*                                       -------------------------
*
*
*                                       -------------------------
*                                       | o     o  o  o     o  o | ON
*  UART Port 3 selection DS4: 0x110     |    o           o       | OFF
*                                       | 1  2  3  4  5  6  7  8 |
*                                       -------------------------
*
*
*                                       -------------------------
*                                       | o     o  o  o        o | ON
*  UART Port 4 selection DS5: 0x118     |    o           o  o    | OFF
*                                       | 1  2  3  4  5  6  7  8 |
*                                       -------------------------
*
*
*                                       -------------------------
*                                       | o     o  o     o  o  o | ON
*  UART Port 5 selection DS6: 0x120     |    o        o          | OFF
*                                       | 1  2  3  4  5  6  7  8 |
*                                       -------------------------
*
*
*                                       -------------------------
*                                       | o     o  o     o     o | ON
*  UART Port 6 selection DS7: 0x128     |    o        o     o    | OFF
```

```
 *                                        | 1  2  3  4  5  6  7  8 |
 *                                        --------------------------
 *
 *                                        --------------------------
 *                                        | o     o  o        o  o | ON
 *  UART Port 7 selection DS8: 0x130      |    o        o  o       | OFF
 *                                        | 1  2  3  4  5  6  7  8 |
 *                                        --------------------------
 *
 *                                        --------------------------
 *                                        | o     o  o           o | ON
 *  UART Port 8 selection DS9: 0x138      |    o        o  o  o    | OFF
 *                                        | 1  2  3  4  5  6  7  8 |
 *                                        --------------------------
 *
 *                                    |-|
 * IRQ line selection:  IRQ3:         |o|   o    o    o    o    o
 *                                    |o|   o    o    o    o    o
 *                                    |-|
 *                                    J85  J86  J87  J88  J89  J90
 *                                    IRQ3 IRQ5 IRQ7 IRQ6 IRQ4 IRQ2
 *
 *
 * Interrupt selection: all ODD: |o| |o| |o| |o| |o| |o| |o| |o|  o   o
 * (Board ID 0)                  | | | | | | | | | | | | | | | |
 *                               |o| |o| |o| |o| |o| |o| |o| |o| |o| |o|
 *                                -   -   -   -   -   -   -   -  | | | |
 *                                o   o   o   o   o   o   o   o  |o| |o|
 *                                                                -   -
 *                                J1  J2  J3  J4  J5  J6  J7  J8  J9  J10
 *                                P1  P2  P3  P4  P5  P6  P7  P8  BOARD ID
 *
 */
int StatusRegisterAddress;   /* These three variables are used to help      */
int SharedInterruptNumber;   /* set the defaults for the multiport boards.  */
int FirstUartAddress;        /* If the board to be tested is not configured */
                             /* the way the setv_XXXX routine expects, the  */
                             /* user needs to override the settings using   */
                             /* the command line switches.                  */

int NextFreePort;          /* NextFreePort is used during the initialization*/
                           /* phase.  Every time a new port is set up using */
                           /* the asisetv routine, its number is determined */
                           /* by NextFreePort.  The variable is then bumped */
                           /* so the next port will have a new number.      */

int MicroChannel;          /* During initialization, this program checks to */
                           /* see if this is a PS/2 with a micro channel    */
                           /* bus.  This variable is set TRUE if it is.     */
```

```
int ActiveChannels;           /* After all the ports are opened with asiopen() */
                              /* statements, this variable is properly set to  */
                              /* the number of ports that were opened up, and  */
                              /* will now be scanned.                          */

char TryToOpen[ MAX_PORT ];  /* Each time a port is set up with an asisetv()*/
char ActivePort[ MAX_PORT ]; /* call, its position in TryToOpen is set TRUE.*/
                             /* Later on, the program tries to open all thes*/
                             /* ports set.  The ones that open successfully */
                             /* are set TRUE in the ActivePort array.       */
#define BREAK_DELAY     3       /* all more or less self-explanatory.        */
#define WTIME           0
#define RTIME           0
#define WMODEM          ON
#define RTS             ON
#define DTR             ON

#define BASE8259        0x20    /* This is the address for the 8259 interrupt*/
                                /* controller.  It is the same for every PC  */
                                /* in existence, so don't even think about   */
                                /* changing it.                              */



int setv_pc_digiboard(int channel_count)
{
    int offset;
    int status;
    int current_uart_address;

     StatusRegisterAddress = 0x140; /* the board are set to the defaults,*/
     SharedInterruptNumber = IRQ5;  //was IRQ3
    current_uart_address = 0x100;
     NextFreePort = COM3;
    asishare( 0x001f );              /* Set up the shared interrupt param */

    for ( offset = 0 ; offset < channel_count ; offset++ )
         {
      status = asisetv( NextFreePort, current_uart_address,
                          SharedInterruptNumber + 8, BASE8259,
                          SharedInterruptNumber, BREAK_DELAY, WMODEM,
                          WTIME, RTIME, StatusRegisterAddress,
                          offset );
      if ( status != ASSUCCESS )
              {
         printf( "pc8_set can't open port COM%d:. Status=%d\n",
                      NextFreePort + 1, status );
         return( status );
```

```
        } else
             {
             }
    TryToOpen[ NextFreePort++ ] = TRUE;
    current_uart_address += 8;
    if ( current_uart_address == 0x140 )      /* These few lines of code */
          current_uart_address = 0x148;         /* cause the UART addresses*/
       else if ( current_uart_address == 0x1a0 )/* to go through the sequen*/
          current_uart_address = 0x208;         /* used by the PC/16 boards*/
    }
  return(1);
}
```

```
//  set_pg:
//  set up the environment of pageit
//

#include <stdlib.h>
#include <signal.h>
#include <stdio.h>

FILE     *f_log;        //file handle for error log ( used by 'log' below)

int set_pg(argc, argv, page_dir)
int argc;
char     **argv;
char     **page_dir;    // directory where sampage lives
{
    *page_dir=argv[1];

    // check the number of arguments
    if (argc == 3) {
         // get the log file name
         //redirect stderr to log file
         f_log=freopen(argv[2],"a",stderr);
         if(f_log==NULL) { //shall indicate no log file
              printf("WARNING - log file open error: %s\n",
                   sys_errlist[errno]);

         }

    }
    else if (argc == 2) {
         f_log=NULL; // no log file
    }
    else {
         return(-1);
    }

return(0);

}

// routine to write a message to the log file.
// if no log file, write to screen.

#include <time.h>
#include <string.h>
#include <stdio.h>

void log(msg)

char *msg;
```

```
{
char date[9]; //system date
char time[9]; //system time


    _strdate(date);
    _strtime(time);

    if(f_log !=NULL){
         if(fprintf(f_log,"%s %s:",date,time)<=0) f_log=NULL;
    }
    if(f_log !=NULL){
         if(fprintf(f_log,"%s\n",msg)<=0) f_log=NULL;
    }

    if(f_log == NULL){ // write to console
         printf("log message: %s %s:",date,time);
         printf(" %s\n",msg);
    }
}
```