

8. FILES

VSE provides the standard file functions that are available in a regular programming language. Files can be created, read and written using standard commands.

8.1 Sequential Files

There are three types of sequential files.

TEXT: Text files can be created using a text editor like Word Pad or Ultra Edit in the Windows environment.

FIXED: Fixed files contain fixed length records. They may contain any form of binary data.

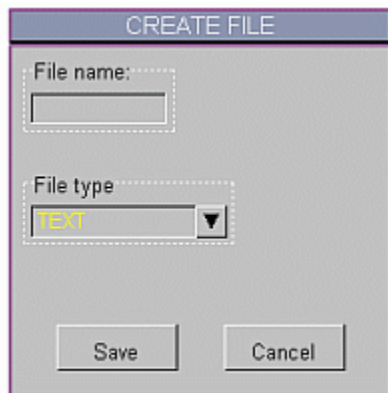
VARIABLE: Variable files contain variable length records. These may contain binary data as well. They can be of two types -

- (a) Automatic, where the system stores the record lengths with each physical record, or
- (b) User Known, where the user knows the length of each record.

8.2 Using Files in a Task



Files are represented by the file icon in a task. The icon is connected to a resource and the resource, in turn, describes the format of the connected file. A record in the file corresponds to a variable declared within the resource. Read and write commands are used to transfer data from the file to the variables and vice versa.

A dialog box titled "CREATE FILE" with a grey background. It contains two input fields: "File name:" with an empty text box, and "File type:" with a dropdown menu showing "TEXT" in yellow. At the bottom, there are two buttons: "Save" and "Cancel".

The file icon can be found on the left column of the interface. As with other VSE elements, it is connected to the resource using a line.

While connecting, you will be prompted for the file name and type.

Resources that interface with a file automatically become *external* and *dedicated*. Such resources can be of two types –

Input resources: These contain input data for the program. These are read by the process that is used to access the file.

Output resources: These contain the output data from the program. These are written to by the processes that access the output file.

8.3 File Statements

The following statements are used in processes which access user files.

OPEN and CLOSE

Before an external file may be used by a process, it must be named in an OPEN statement. For sequential files, the OPEN command is used in conjunction with the word INPUT or OUTPUT for reading or writing to the file.

The CLOSE statement terminates the effect of the most recent OPEN statement for a specific named resource. Its effect is to close the external file.

No READ or WRITE statements may be used for this resource once the CLOSE statement is written. Each OPEN statement, however, must be paired with a CLOSE statement.

E.g. `OPEN INPUT INPUT_MESSAGE`
`OPEN OUTPUT TRANSMISSION`
`OPEN INPUT_OUTPUT NEEDLINE_FILE`
`CLOSE MY_RESOURCE`

READ

This statement enables data records to be read from an external file. It may only be used in a process which interfaces with that file. It copies the next sequential record from the external file into the named resource.

```
READ resource_name AT_END statement
```

The AT_END clause must be used to control what happens when the process attempts to read past the end of the external file. Only the first simple statement following the

AT_END keyword will be in the AT_END clause, and this cannot be a conditional statement.

e.g.

```
READ INPUT_MESSAGE
    AT_END EXECUTE CLOSE_DOWN
READ EXTERNAL_FILE
    AT_END EXECUTE SYNTAX_CHECK
READ EDITOR_FILE WITH RECORD_LENGTH = SIZE
    AT_END SET FILE_STATE TO END_OF_FILE
```

WRITE

This statement enables data records to be written to an external file. It may only be used in a process which interfaces with that file. It copies the named resource into the next sequential record for the external file.

```
WRITE resource-name [[WITH] SIZE = size_attribute] [WITHOUT LINEFEED]
```

e.g.

```
WRITE OUTPUT_MESSAGE
WRITE TRANSMISSION_RECORD WITH SIZE = MESSAGE_LENGTH
WRITE PRINTER_RECORD WITH SIZE = ESCAPE_SEQUENCE_SIZE
WITHOUT LINEFEED
```

FILE

The file condition can be used to determine whether a file already exists before creating new ones. It can also determine whether an existing file is empty.

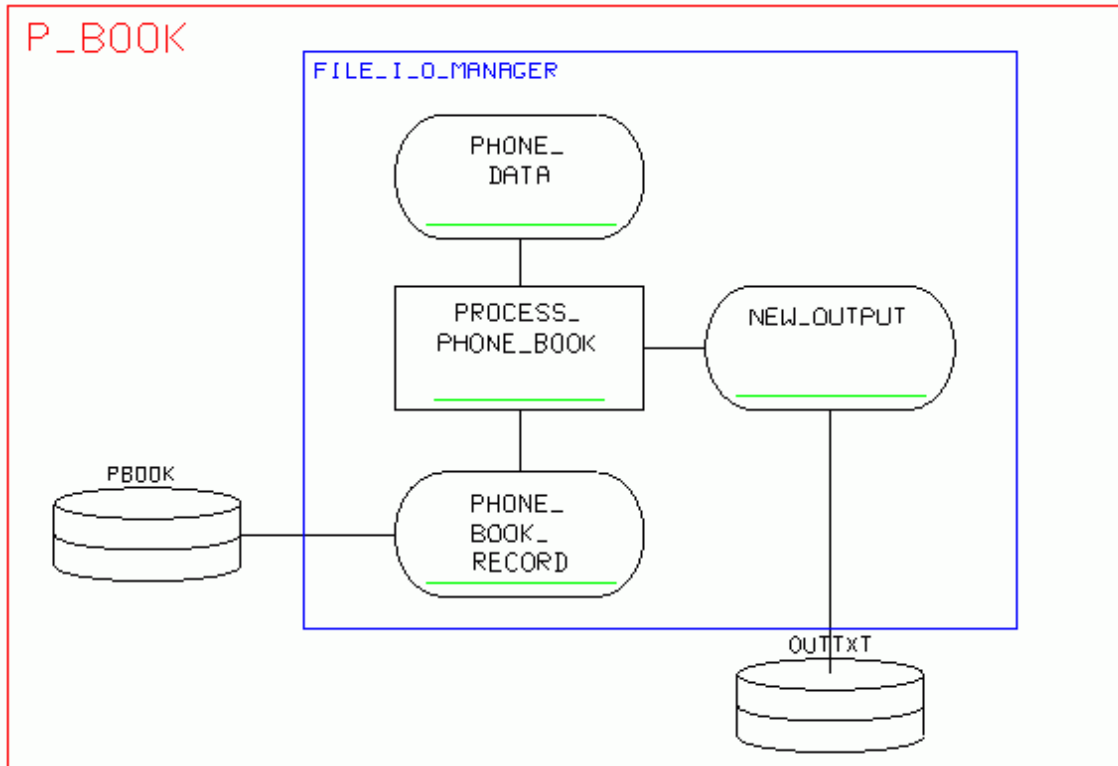
e.g.

```
IF MY_FILE EXISTS
AND MY_FILE IS NOT EMPTY
    ASSIGN MY_FILE TO MY_RESOURCE.
```

8.4 Example Program – Phone Book Manager

This program reads entries from an input file, which contains a series of names and phone numbers, and copies them to an output file. It also prints the entries to the console.

The module contains a process and resource, along with an additional resource that is used to interface with a file. When you are done, your task should look similar to the following figure.



The PHONE_DATA resource contains the following code

```

***PHONE_DATA

PHONE_BOOK_STATE    STATUS    CLOSED
                   END_OF_FILE
                   OPEN_FOR_READ
                   OPEN_FOR_WRITE
                   INITIAL_VALUE CLOSED

NUMBER_OF_RECORDS   INDEX    INITIAL_VALUE 0
RECORD_NO           INDEX

PHONE_BOOK_RECORD   QUANTITY(100) CHAR 45
  
```

The PHONE_BOOK_RECORD resource contains the records that will be filled with data read from the input file.

```

***PHONE_BOOK_RECORD

PHONE_BOOK_ENTRY
  1  NAME
    2  FIRST_NAME          CHAR 10
    2  MIDDLE_INITIAL      CHAR 3
    2  LAST_NAME           CHAR 10
  
```

```

1  PHONE_NUMBER
2  AREA_CODE           CHAR 3
2  LOCAL_NUMBER       CHAR 7
2  EXTENSION          CHAR 5

```

The NEW_OUTPUT resource is used to connect the output file to the process.

```

NEW_OUTPUT
PAD      CHAR 100

```

The process PROCESS_PHONE_BOOK will contain the following code.

```

PROCESS_PHONE_BOOK
  EXECUTE READ_IN_PHONE_BOOK
  EXECUTE PRINT_PHONE_BOOK

*** DATA IS READ FROM THE INPUT FILE

READ_IN_PHONE_BOOK
  OPEN INPUT_PHONE_BOOK
  SET PHONE_BOOK_STATE TO OPEN_FOR_READ
  EXECUTE READ_PHONE_RECORD
    UNTIL PHONE_BOOK_STATE IS END_OF_FILE
  CLOSE PHONE_BOOK

READ_PHONE_RECORD
  READ_PHONE_BOOK
  AT_END SET PHONE_BOOK_STATE TO END_OF_FILE

  IF PHONE_BOOK_STATE IS END_OF_FILE
    EXIT THIS RULE.

  INCREMENT NUMBER_OF_RECORDS
  MOVE PHONE_BOOK_ENTRY
    TO PHONE_BOOK_RECORD(NUMBER_OF_RECORDS)

  DISPLAY_PHONE_BOOK_ENTRY

*** DATA IS WRITTEN TO THE OUTPUT FILE

PRINT_PHONE_BOOK
  OPEN_OUTPUT_PHONE_BOOK
  OPEN_OUTPUT_NEW_OUTPUT
  SET_PHONE_BOOK_STATE TO OPEN_FOR_WRITE

  EXECUTE_PRINT_PHONE_RECORD
    INCREMENTING_RECORD_NO
    UNTIL RECORD_NO IS GREATER THAN NUMBER_OF_RECORDS

  CLOSE_PHONE_BOOK
  CLOSE_NEW_OUTPUT

PRINT_PHONE_RECORD
  MOVE_PHONE_BOOK_RECORD(RECORD_NO)

```

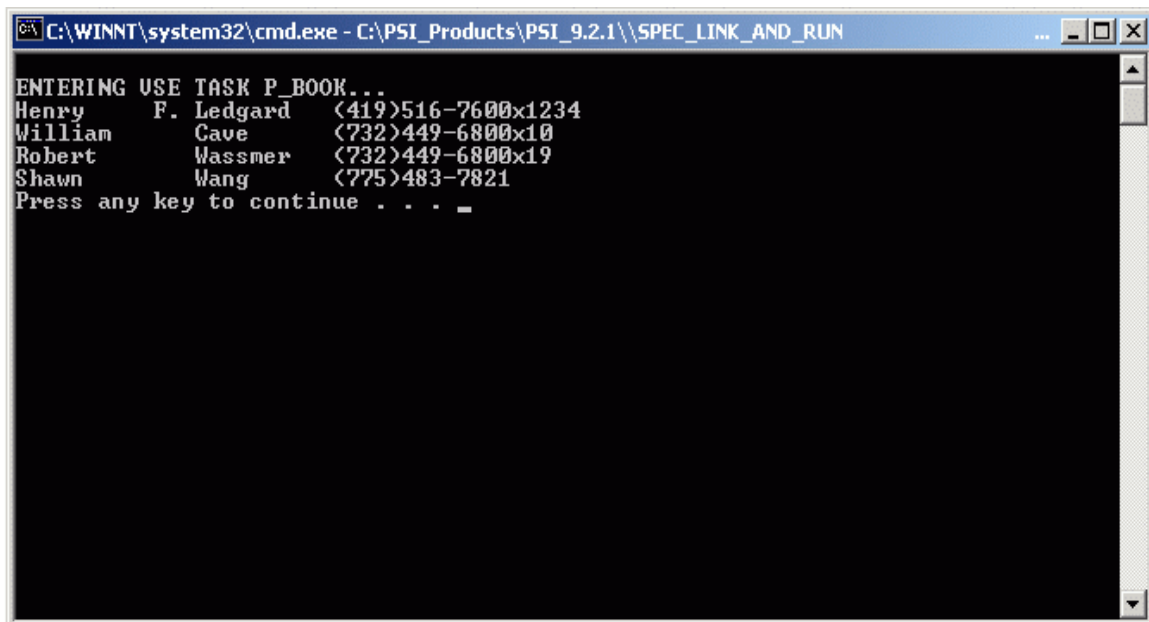
```
TO PHONE_BOOK_ENTRY
MOVE PHONE_BOOK_RECORD (RECORD_NO)
TO NEW_OUTPUT
```

```
WRITE PHONE_BOOK_RECORD
WRITE NEW_OUTPUT
```

Double click on the PBOOK file icon and enter some input data into the editor. The input might look like this.

```
Henry      F. Ledgard   (419) 516-7600x1234
William    Cave        (732) 449-6800x10
Robert     Wassmer     (732) 449-6800x19
Shawn      Wang        (775) 483-7821
```

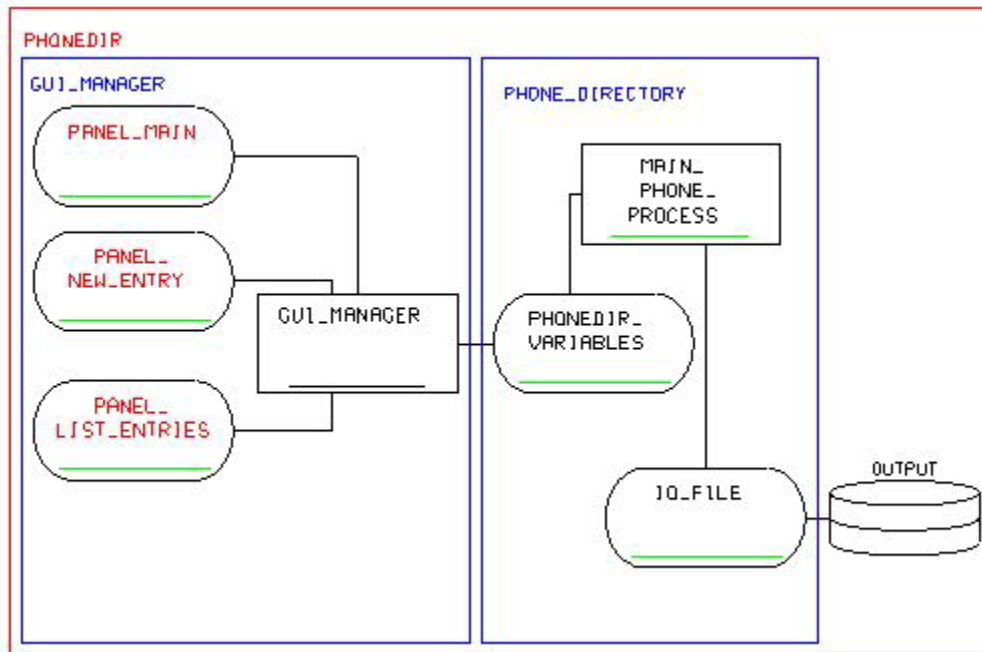
Prepare the task and run it. The input data will be printed to the console. You will find that the data has been written to the output file as well. Double click the NEW_OUTPUT icon to verify this.



```
C:\WINNT\system32\cmd.exe - C:\PSI_Products\PSI_9.2.1\SPEC_LINK_AND_RUN
ENTERING USE TASK P_BOOK...
Henry      F. Ledgard   <419>516-7600x1234
William    Cave        <732>449-6800x10
Robert     Wassmer     <732>449-6800x19
Shawn      Wang        <775>483-7821
Press any key to continue . . . _
```

8.6 Phone Book Manager using a PLM Interface

The above program is now implemented using a GUI interface. The program uses two modules to separate the functionality.



The GUI_INTERFACE module manages the various panels used and the PHONE_DIRECTORY module carries out the file input output operations.

MODULE: GUI_INTERFACE

The three panels are created in the PLM as shown:

PANEL_MAIN

PHONE BOOK

NEW ENTRY

LIST ENTRIES

EXIT

PANEL_ADD

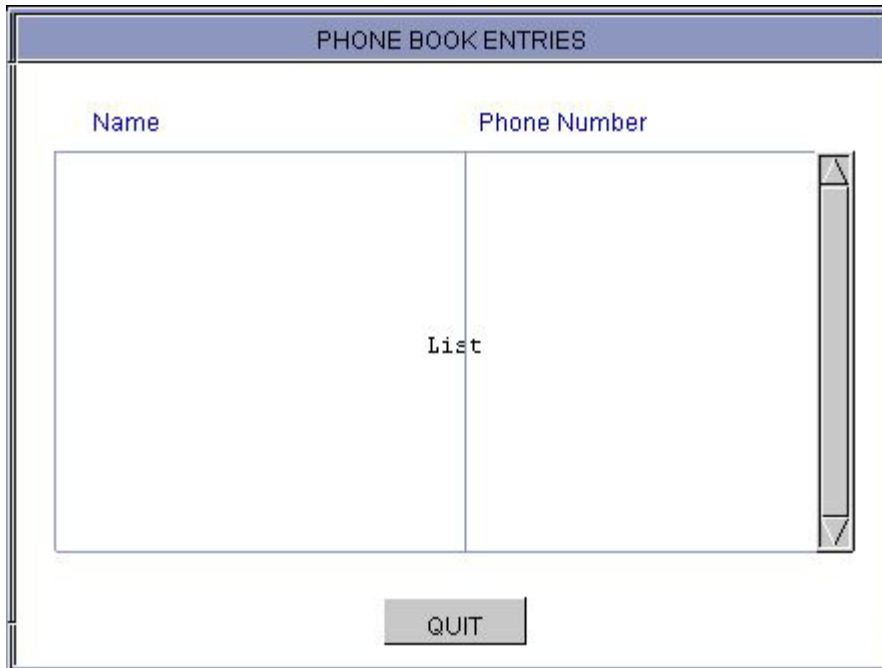
ENTER NEW ENTRY

NAME

PHONE NUMBER

SAVE QUIT

PANEL_LIST



The code for these panels is automatically generated by VSE.

GUI_MANAGER

```

GUI_MANAGER
  EXECUTE PROCESS_EVENTS
  UNTIL QUIT_MAIN_FORM PANEL_BUTTON_STATUS IS ON

PROCESS_EVENTS
  DISPLAY PANEL PANEL_MAIN
  ACCEPT PANEL PANEL_MAIN
  REMOVE PANEL PANEL_MAIN

  IF RTG_PANEL_WIDGET_NAME IS EQUAL TO 'NEW_ENTRY'
    SET NEW_ENTRY PANEL_BUTTON_STATUS TO OFF
    EXECUTE ENTRY_MENU
  ELSE IF RTG_PANEL_WIDGET_NAME IS EQUAL TO 'LIST_ENTRIES'
    SET LIST_ENTRIES PANEL_BUTTON_STATUS TO OFF
    EXECUTE LIST_MENU .

ENTRY_MENU
  EXECUTE INITIALIZE_ENTRY_FORM
  DISPLAY PANEL PANEL_NEW_ENTRY

```

```

EXECUTE PROCESS_ENTRY_CMDS
  UNTIL QUIT_ENTRY_FORM PANEL_BUTTON_STATUS IS ON

REMOVE  PANEL PANEL_NEW_ENTRY

PROCESS_ENTRY_CMDS
ACCEPT PANEL PANEL_NEW_ENTRY

IF SAVE_ENTRY PANEL_BUTTON_STATUS IS ON
  EXECUTE ENTER_DATA .

DISPLAY PANEL PANEL_NEW_ENTRY

INITIALIZE_ENTRY_FORM
MOVE SPACES TO INPUT_NAME  PANEL_INPUT_TEXT
MOVE SPACES TO INPUT_NUMBER PANEL_INPUT_TEXT
SET QUIT_ENTRY_FORM PANEL_BUTTON_STATUS TO OFF
SET SAVE_ENTRY PANEL_BUTTON_STATUS TO OFF

ENTER_DATA
IF INPUT_NAME PANEL_INPUT_TEXT IS EQUAL TO SPACES
  MOVE 'Name field cannot be empty!'
  TO NEW_ENTRY_MESSAGE PANEL_TEXT
  SET SAVE_ENTRY PANEL_BUTTON_STATUS TO OFF
  EXIT THIS RULE .

INCREMENT NUMBER_OF_ENTRIES
MOVE INPUT_NAME  PANEL_INPUT_TEXT
  TO ENTRY_NAME (NUMBER_OF_ENTRIES)
MOVE INPUT_NUMBER PANEL_INPUT_TEXT
  TO ENTRY_NUMBER (NUMBER_OF_ENTRIES)
EXECUTE INITIALIZE_ENTRY_FORM
MOVE 'Enter new entry, or click QUIT to exit.'
  TO NEW_ENTRY_MESSAGE PANEL_TEXT

LIST_MENU
MOVE SPACES TO SCROLL_LIST_ENTRIES
EXECUTE INITIALIZE_SCROLL_LIST
  INCREMENTING ENTRY_NUMBER
  UNTIL ENTRY_NUMBER IS GREATER THAN NUMBER_OF_ENTRIES
PANEL_SCROLL_LIST_SELECT = 1
SET QUIT_LIST_FORM PANEL_BUTTON_STATUS TO OFF

DISPLAY PANEL PANEL_LIST_ENTRIES

EXECUTE PROCESS_LIST_CMDS
  UNTIL QUIT_LIST_FORM PANEL_BUTTON_STATUS IS ON

REMOVE  PANEL PANEL_LIST_ENTRIES

INITIALIZE_SCROLL_LIST
MOVE ENTRY (ENTRY_NUMBER) TO PANEL_SCROLL_TEXT (ENTRY_NUMBER)

PROCESS_LIST_CMDS
ACCEPT  PANEL PANEL_LIST_ENTRIES

```

MODULE: PHONE_DIRECTORY

PHONEDIR_VARIABLES

```
FILE_STATE      STATUS  CLOSED
                                EOF
                                OPEN_READ
                                OPEN_WRITE
                                INITIAL_VALUE CLOSED

NUMBER_OF_ENTRIES  INDEX  INITIAL_VALUE 0
ENTRY_NUMBER       INDEX

PHONE_BOOK
  1 ENTRY          QUANTITY(100)
  2 NAME           CHAR 20
  2 PAD            CHAR 4
  2 NUMBER         CHAR 13
```

IO_FILE

```
FILE_ENTRY
  1 NAME           CHAR 20
  1 PAD            CHAR 4
  1 NUMBER         CHAR 13
```

MAIN_PHONE_PROCESS

```
PROCESS_FILE
  EXECUTE READ_FILE
  CALL GUI_MANAGER
  EXECUTE WRITE_FILE

READ_FILE
  MOVE SPACES TO PHONE_BOOK
  ASSIGN 'PHONEBOOK.TXT' TO IO_FILE
  IF 'PHONEBOOK.TXT' DOES NOT EXIST OR
    'PHONEBOOK.TXT' IS EMPTY
    EXIT THIS RULE .

  OPEN INPUT IO_FILE
  SET FILE_STATE TO OPEN_READ
  EXECUTE READ_ENTRIES
    UNTIL FILE_STATE IS EOF
  CLOSE IO_FILE

READ_ENTRIES
  READ IO_FILE
```

```

        AT_END EXECUTE SET_EOF
    IF FILE_STATE IS EOF
        EXIT THIS RULE.
    INCREMENT NUMBER_OF_ENTRIES
    MOVE FILE_ENTRY TO ENTRY(NUMBER_OF_ENTRIES)
    DISPLAY FILE_ENTRY, NUMBER_OF_ENTRIES
    ***DISPLAY NUMBER_OF_ENTRIES

SET_EOF
    SET FILE_STATE TO EOF

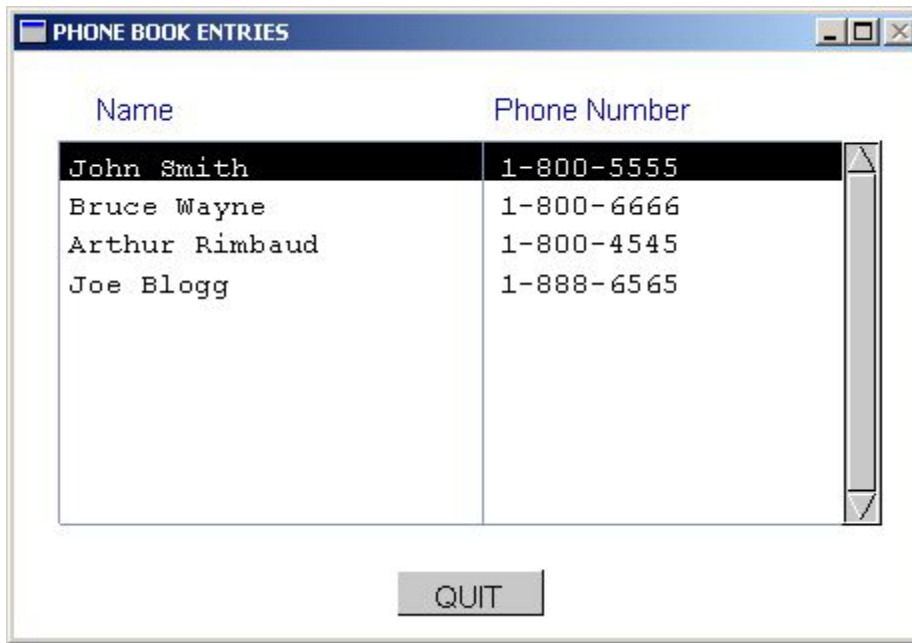
WRITE_FILE
    OPEN OUTPUT IO_FILE
    SET FILE_STATE TO OPEN_WRITE
    EXECUTE WRITE_ENTRIES INCREMENTING ENTRY_NUMBER
        UNTIL ENTRY_NUMBER IS GREATER THAN NUMBER_OF_ENTRIES
    CLOSE IO_FILE

WRITE_ENTRIES
    MOVE ENTRY(ENTRY_NUMBER) TO FILE_ENTRY
    DISPLAY 'WRITING TO OUTPUT FILE:', ENTRY_NUMBER,
        ' ', FILE_ENTRY
    WRITE IO_FILE

```

The program is prepared and run as usual. A file called OUTPUT.TXT has to be created before running the program. This file will be used to store the phone book entries.

The image shows a graphical user interface window titled "ENTER NEW ENTRY". It features two text input fields. The first field, labeled "NAME", contains the text "John Smith". The second field, labeled "PHONE NUMBER", contains the text "1-800-5555". Below these fields are two buttons: "SAVE" and "QUIT". The window has a standard Windows-style title bar with minimize, maximize, and close buttons.



The entries will also be stored in the OUTPUT.TXT file.
