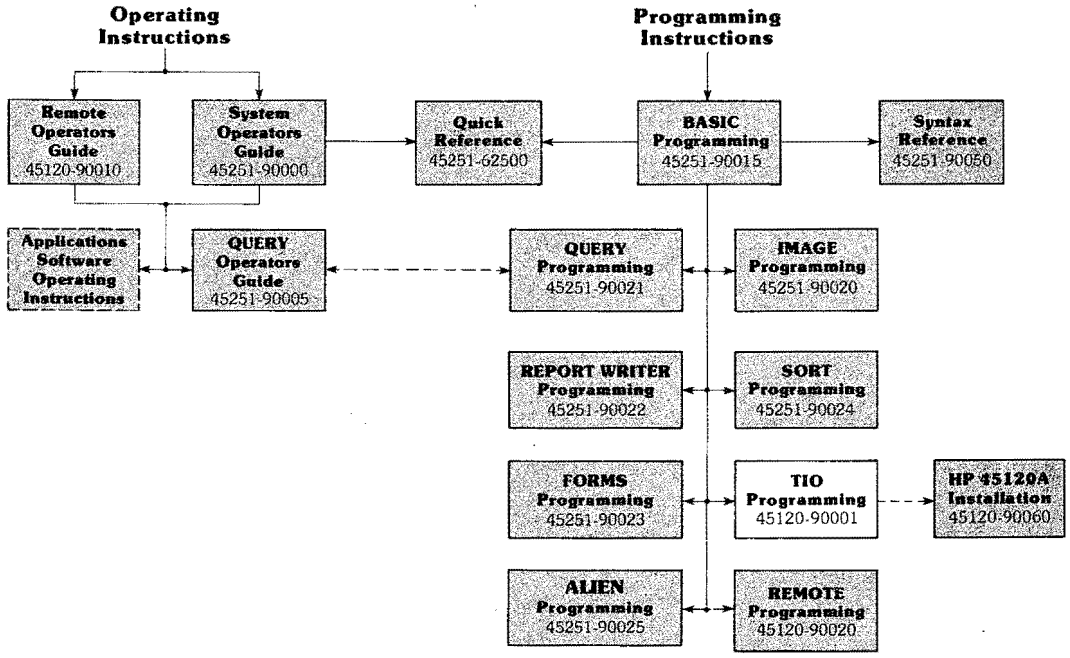
TIO/250 Programming Manual

HP 250



HP 250 DOCUMENTATION SCHEME

TIO/250
Programming Manual

Manual Part No. 45120-90001

Hewlett-Packard
General Systems Division
19447 Pruneridge Ave., Cupertino, California 95014
(For World-wide Sales and Service Offices see back of manual.)

Printing History

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revised date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

February 1979 ... **FIRST EDITION**
June 1979 ... Update for O.S. 2D: pages C-1 thru C-10
December 1979 ... **SECOND EDITION**

Table of Contents

CHAPTER 1
Introduction

The TIO DROM (Disc Resident Optional Module) and the HP45120A Asynchronous Serial Interface (ASI) provide a means to connect up to five additional RS-232 asynchronous devices to the HP 250. The devices may be terminals, printers or HP 3000 computers.

An application program can be written to send output to and receive input from the devices. TIO statements are available to determine the state of each connection (port). These statements are described in Chapter 2.

The TIO DROM is supplied on HP 250 Operating System discs beginning with revision 1C.

Prerequisite Reading

This manual assumes that the reader is familiar with the HP 250 and has used and understood the BASIC Programming Manual. It is also assumed that the manuals which accompany the remote printers and terminals have been read, and the reader understands the devices. Use of the HP 3000 requires a thorough understanding of that computer's operating system.

Introduction to TIO

A port connecting a printer to the HP 250 is used for output. When the ASI output buffer for the port is available, a PRINT or PRINT USING statement is used to send output data.

A port connecting a terminal to the HP 250 is used for both input and output. The program reads data from the port's ASI input buffer. Then the remote device sends more data. The program can send output when the port's ASI output buffer is empty. A terminal operates in half-duplex fashion; it is ready for either input or output at any one time. The program determines which one.

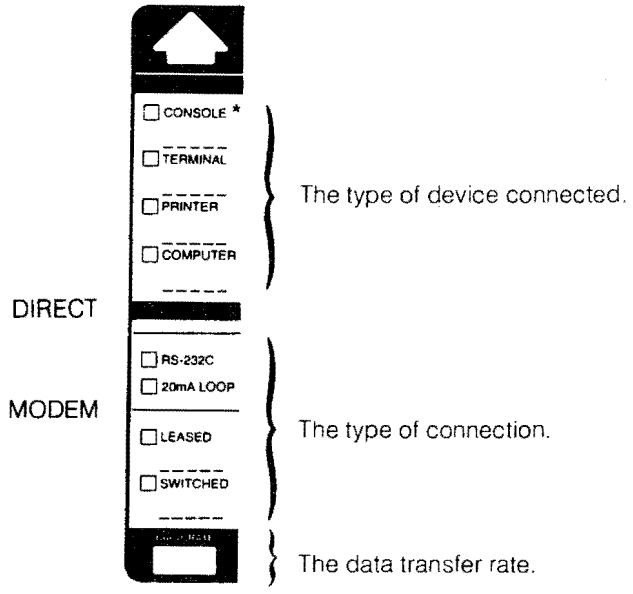
The port for a computer is always ready for input from the HP 3000 and can simultaneously be ready for output from the program.

The ASI Installation Note (part no. 45120-90065) describes the hardware configuration necessary to connect ports to the various devices. Configuring the TIO software into the system is described in Appendix C of the BASIC/250 Programming Manual.

The program uses PRINTER IS, PRINT ALL IS or SYSTEM PRINTER IS to direct output to a remote device. The device connected to port 1 has a device address of 11; the device connected to port 2 has a device address of 12, etc.

The TIO DROM keeps status information on each port (is the device connected? disconnected? is the input/output buffer full/empty? etc.). This information is transferred to the program through interrupts if the program requests it. The program requests the interrupt through the use of the TIO statements.

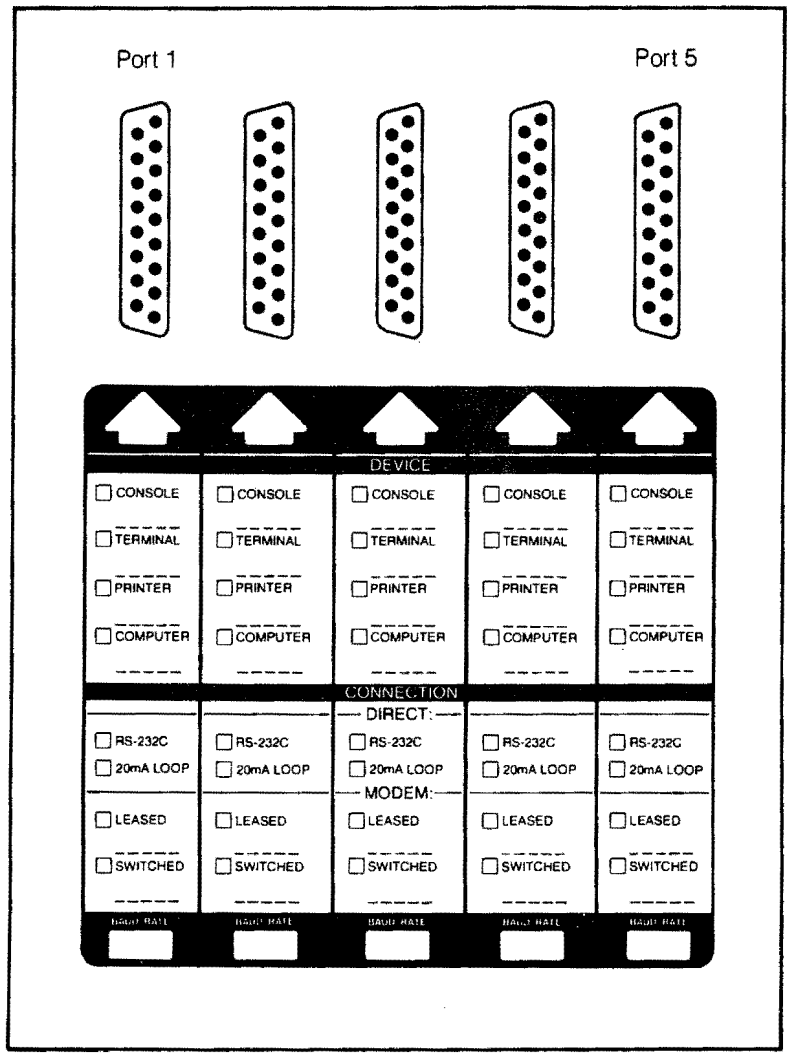
The cables which connect the HP 250 with the remote device plug into a panel on the side of the tub. Each connection has a table which is filled in during the installation of TIO.



*The Remote Console is not available at this time.

ASI Connection Table

Some TIO commands are used exclusively for the HP 3000 computer. Other commands are used for the terminal or printer. You need to know which device is connected to which port. The ports are numbered 1 to 5 from left to right. (The device address is the port number plus 10.)



Port Numbers

CHAPTER 2
TIO Statements

The ON-condition Statements

The TIO statements operate similar to the ON KEY # statement*, in that the program continues execution until a preset condition occurs. When the condition occurs, the program branches to a specified routine.

The syntax of the "ON" statements is as follows:

ON condition # device address [,priority] branching statement

The condition can be one of the following:

BREAK	When a break is received, the interrupt occurs.
CONNECT	When the remote device is connected, the interrupt occurs.
DISCONNECT	When the remote device is disconnected, the interrupt occurs.
INPUT	When an input terminator (CR) is received, the interrupt occurs.
OUTPUT	When the output buffer is empty, the interrupt occurs.
TRIGGER	When a DC1 is received from the HP 3000 the interrupt occurs.

The device address is the port number plus 10; e.g., the device attached to port 3 has a device address of 13.

Before an ON-condition statement is executed, the program must have exclusive use of the device. This is done with the REQUEST statement which is described later in this chapter.

The priority parameter is used by the system to determine which interrupt to service first. The priorities range from 1 - the lowest, thru 15 - the highest.

A priority of 1 is assumed when the parameter is not used. The first interrupt condition to occur with a priority greater than the current execution priority will be processed.

* ON KEY # is described in the Branching and Subroutines chapter of the BASIC Programming Manual.

The branching statement may be a GOTO, GOSUB or CALL statement. If the branch is a GOTO statement, the execution priority remains the same as before the interrupt occurred. If the branch is a GOSUB or CALL statement, the execution priority changes to the priority specified in the ON statement until the end of the subroutine or subprogram. It is then restored to the execution priority in effect before the interrupt occurred.

GOTO — When an interrupt occurs with the sufficient priority, the program branches to the specified line and continues execution. The program cannot return to the line where the interrupt occurred. The GOTO statement uses less system overhead than the GOSUB or CALL statements use.

GOSUB — When an interrupt occurs, the program branches to the subroutine. After the subroutine has been executed, the program returns to the line where the interrupt occurred.

CALL — When an interrupt occurs, the program branches to the subprogram. After the subprogram has been executed, the program returns to the line where the interrupt occurred. Parameters cannot be passed.

Subprograms create a new environment during their execution. If an interrupt condition occurs during a subprogram and the action is another CALL with priority higher than the current execution priority, then the interrupt is serviced. If the action is a GOTO or GOSUB, the interrupt is not serviced until the subprogram exits with a SUBEXIT or SUBEND.

An interrupt generally occurs after execution of the current line. However, if an interrupt occurs during execution of a WAIT or INPUT statement, execution of the statement is suspended while the interrupt is serviced.

To prevent interrupts during critical portions of a program, use DISABLE and ENABLE statements to enclose the lines. Once the DISABLE statement is executed, no interrupts occur until the ENABLE statement is executed.

The ON BREAK # Statement

The HP 264x series terminals have a key labeled BREAK. If this key is pressed, the ON BREAK # statement will detect it.

ON BREAK # device address [,priority] branching statement

If the break occurs when the port is in the output state, the remainder of the current output buffer is discarded, the break interrupt is activated and the port remains in the output state. The interrupt routine specified in the ON BREAK # statement should output a prompt to the remote terminal user to indicate the break has been serviced. The interrupt routine would typically enable input from the terminal through the ON INPUT # statement in order to determine the operation desired by the user.

If the break occurs when the port is in the input state, then the input state is cancelled and the port is set in the output state. The input buffer is discarded. The interrupt routine may then output a prompt as outlined above.

The ON BREAK # condition is cancelled by executing an OFF BREAK # statement.

OFF BREAK # device address

If the program has not established a break interrupt routine, or has cancelled the condition by execution of an OFF BREAK # statement, pressing the BREAK key on the remote terminal while the port is in output mode has no effect. Pressing the BREAK key while in the input mode, however, causes the current input line to be rejected (as if the remote terminal user pressed control-X) and a new input line is begun.

Examples:

```
ON BREAK #Terminal,5 GOTO Break

OFF BREAK #Terminal

ON BREAK #14 CALL Break

ON BREAK #Terminal+10,Priority GOSUB Break
```

The ON CONNECT/DISCONNECT # Statements

The program can detect if a device has been connected or disconnected by use of the ON CONNECT # or ON DISCONNECT # statements.

ON CONNECT # device address [, priority] branching statement

ON DISCONNECT # device address [, priority] branching statement

Only one of these statements may be in effect at any time. The ON CONNECT # statement cancels execution of the ON DISCONNECT # statement. And the ON DISCONNECT # statement cancels the ON CONNECT # statement. The ON DISCONNECT # statement should be executed at the beginning of program execution, or when the device is first addressed. If the port is actually disconnected, then the interrupt condition is satisfied immediately. The ON CONNECT # statement should then be executed in response to the disconnect interrupt. If the port is connected, then the interrupt condition is satisfied and the device can be used.

NOTE

These interrupts are serviced according to the hardware jumpers' configuration on the connection board. Configuration is described in the HP 45120A Installation Note.

The program may terminate detection of the connect/disconnect interrupt condition by use of the corresponding OFF CONNECT # or OFF DISCONNECT # statement:

OFF CONNECT # device address

OFF DISCONNECT # device address

Examples:

```
FOR Device=1 TO 5
    ON DISCONNECT #Device+10 CALL Disc
NEXT Device

ON CONNECT #12 GOTO Int_12

ON CONNECT #14,Priority GOSUB Connect

OFF CONNECT #Device

OFF DISCONNECT #14
```

The ON INPUT # Statement

The ON INPUT # statement explicitly enables input from a terminal and informs the system of the desired action to be taken when a complete input line is received.

ON INPUT # device address [, priority] [branching statement]

The ON INPUT # statement remains in effect until input is received, an ON OUTPUT # statement addressing the same port is executed or an OFF INPUT # statement is executed.

If the branching statement is omitted, TIO assumes that an ON INPUT # statement specifying a branch was previously executed.

For example, assume the subroutine Txy is called whenever input is available. The first ON INPUT # statement would have a branching statement of GOSUB Txy. When input is received from the terminal an interrupt occurs and program execution continues in the Txy subroutine. Before the subroutine is ended with a RETURN, the ON INPUT # statement with no branch is executed. When input is received from the terminal, an interrupt occurs and the action statement in the previous ON INPUT # statement, GOSUB Txy, is executed.

As the terminal user enters a line of information, the ASI echoes each character as it is accepted and stores it in the input buffer associated with the port. The input buffer holds up to 255 characters. The user may backspace and retype one or more characters by means of the ASCII backspace character, CTRL (H), or the backspace key (not the editing keys). The user may also cancel the entire line and retype it from the beginning by means of the ASCII cancel character CTRL (X). The ASI updates the port input buffer accordingly so the HP 250 is not concerned with the editing operations.

The remote terminal user indicates the end of the entry by a carriage return. The ASI echoes the carriage return, terminates the input state of the remote terminal and notifies the HP 250 of the available input. TIO informs the program of the available input data through a software interrupt. The branching statement is then executed and the application program accepts the input line for processing.

The AREAD$ Function

The string function AREAD$ transfers data from the ASI input buffer to a string variable.

variable$ = AREAD$ (device address)

If the AREAD$ function is attempted when no terminal input line is present in the ASI input buffer, execution error 315, "NO INPUT AVAILABLE", results. The carriage return character is not transferred to the string variable.

When the program has completed processing the input, the program explicitly re-enables further terminal input by means of the ON INPUT # statement.

Examples:

```
        REQUEST 12,Wait
        IF Wait=1 THEN Queue
        ON INPUT #12,3 GOSUB In_12
          .
          .
In_12:  Next_line$=AREAD$(12)
          .
          .
        RELEASE 12
        RETURN
Queue: !


        REQUEST 12
        ON INPUT #12 GOSUB Input
          .
          .
Input:    Next_line$=AREAD$(12)
        IF Next_line$="EXIT" THEN 550
          .
          .
530     ON INPUT #12
540     RETURN
550     RELEASE 12
560     RETURN
```

Cancelling Input

Unusual conditions may arise after a program has enabled input from a remote terminal by means of an ON INPUT # statement, but before the terminal user has supplied a carriage return. For example, the remote user needs to be informed of some pending event (such as the HP 250 is about to be turned off). These conditions require the capability for the program to terminate the input state in order to perform output.

OFF INPUT # device address

Once the OFF INPUT # statement is executed, any input characters already received by the ASI will be discarded. Even if the terminal user has supplied a carriage return, if the input available interrupt is being held pending by the operating system due to execution priority considerations, the entire input line is discarded. The execution of an OFF INPUT # statement addressing a port not currently in the input state results in no operation and is not an error.

Examples:

```
OFF INPUT #12

OFF INPUT #Device
```

Echoing Input

As each character is entered from a terminal, the ASI places it in the ports input buffer and then echoes the character back to the terminal. Some operations, such as data entry from a CRT form or data cartridge, require suppression of input character echoing.

ECHO OFF # device address

ECHO ON # device address

The default mode, established by successful execution of a REQUEST statement, is input echoing on.

If input echoing is disabled, then the ASI does not perform the normal character and line editing functions (backspace and cancel line). Any CTRL (H) or CTRL (X) characters received with echo disabled are stored in the input buffer.

Examples:

```
IF Flag=1 THEN ECHO OFF #Terminal

ECHO ON #13
```

The ON OUTPUT # Statement

The program can direct output to a terminal or remote printer using the PRINTER IS, SYSTEM PRINTER IS and PRINT ALL IS, statements.

When the program is communicating interactively with more than one remote device, the programmer must be concerned with output buffer overflow. The ASI output buffer holds up to 255 data bytes, including carriage return and line feeds. If the output buffer overflows, then the task is blocked until the buffer becomes empty. Unless the program logic considers this possibility, the generation of output for one device may temporarily prevent processing of input and generation of output to all other devices.

The ON OUTPUT # statement detects when the output buffer is empty.

ON OUTPUT # device address [, priority] [branching statement]

The ON OUTPUT # statement remains in effect until an ON INPUT # statement is executed, another ON OUTPUT # statement is executed or an OFF OUTPUT # statement is executed.

The action routine would normally contain one of the print statements which directs output to a particular device. Then any statement which causes output (PRINT, PRINT USING, CAT, etc.) can be used. If the branching statement is omitted, TIO assumes that an ON OUTPUT # statement specifying a branch was executed previously.

For example, assume the main program executes an ON OUTPUT # statement with a branch of GOSUB Txy. When the ASI output buffer for the port is empty, an interrupt occurs and program execution continues in the Txy subroutine. Before the subroutine is ended with a RETURN, the ON OUTPUT # statement with no action statement is executed. When the ASI output buffer is empty, an interrupt occurs and the branch in the previous ON OUTPUT # statement, GOSUB Txy, is executed.

Here are two more examples:

```
      ON OUTPUT #Printer,4 CALL Out_print
        .
        .
SUB Out_print
      Printer=11+(CURKEY-25)/3
      Wait=1
      REQUEST Printer,Wait
      IF Wait=1 THEN GOSUB Queue
      PRINTER IS Printer
      PRINT "Next output"
        .
        .
SUBEND

      ON OUTPUT #12 GOTO Output
        .
        .
Output:   PRINTER IS 12
        .
        .
      ON OUTPUT #12
      RETURN
```

HP 3000 Input/Output

TIO allows a program to communicate with an HP 3000 computer system. A small utility program which emulates an interactive terminal to the HP 3000 exists on the system disc. An HP 3000 to HP 250 file transfer utility also exists. Refer to Appendix C of this manual for details.

Input

The program does not explicitly enable input since the HP 3000 may transmit data at any time. The ON INPUT # statement establishes an interrupt routine which is activated whenever an input line from the HP 3000 is available in the ASI input buffer. The interrupt routine reads the input line by means of the AREAD$ function.

The ASI input buffer for an HP 3000 port is capable of stacking input lines, since the HP 3000 is always enabled for input and the program may not be able to accept and process one input line before the next one arrives. An ENQ/ACK protocol is used to prevent the ASI input buffer from overflowing. (The HP 3000 sends an ENQ [enquiry] after every 80 characters. The HP 250 sends back an ACK [acknowledgement] if there is room for at least an additional 80 characters.)

The convention of ENQ/ACK protocol limits the maximum useful length of each input line to 165 bytes. By disabling the ENQ/ACK protocol (possibly by logging on to the HP 3000 with TERM = 9 for example), the full 255 bytes are available (with some risk of input buffer overflow).

Since HP 3000 input is always enabled, the ON INPUT # statement need not be executed following processing of each line.

The HP 3000 terminates input with either a carriage return or ASCII DC1 (device control one). The carriage return (if any) is included as the last character of the string returned by the AREAD$ function. The DC1 terminator is not included in the input string.

Ready for Output

The HP 3000 must explicitly enable the port for output from the HP 250. To do this, the HP 3000 terminates its data transmission with a DC1. The ON TRIGGER # statement detects the ready state.

ON TRIGGER # device address [,priority] branching statement

Execution of the ON TRIGGER # statement informs the system of the action to be taken upon receipt of the DC1 from the HP 3000. The interrupt is activated only after the program has accepted all the data in the ASI input buffer which was received from the HP 3000 prior to the DC1.

TIO sets the state of the port to output when the interrupt is activated. The ON TRIGGER # branching statement is then executed. The ON TRIGGER # statement remains in effect and does not need to be re-executed after each interrupt.

The interrupt routine uses the PRINTER IS statement to direct output to the HP 3000 port. Only one line of output may be sent to the HP 3000. Output is sent using a PRINT or PRINT USING statement. (ON OUTPUT # is not used.) Once a carriage return has been transmitted, TIO terminates the output state of the port.

Only the output state of the port is affected by the ON TRIGGER # and an interrupt routine. The port is always enabled for input.

Example:

```
1000        ON TRIGGER #12 GOTO Out12
1001 !
2940 Out12:  ON ERROR GOTO Out13
2950        FOR I=1 TO LEN(C$)          !Send command string to computer
2960          PRINT C$[I;1]             !character-by-character.
2970          WAIT 24000/Baud(Baud)     !Delay to allow character turnaround.
2980        NEXT I
2990        PRINT Cr$;                  !terminate the command string.
3000        OFF ERROR
3010        IF Debug AND (State()1) THEN LDISP CHR$(137)&"COMMAND"&
FNState$&CHR$(128)&" "&C$
3020        Command=LEN(C$)             !Flag command sent --- the command
3030        ENABLE                      !string will be echoed and must be
3040        WAIT                        !ignored when received.
3050 !
```

Block Mode

Certain applications, such as file transfers from the HP 250, require the link from the HP 250 to the HP 3000 to use the full output data rate. TIO provides a means to simulate the Block Mode Transfer feature of the HP 264x terminals. This mode ensures reliability of data transfers at the full data rate.

BLOCK MODE ON # device address

BLOCK MODE OFF # device address

After execution of the BLOCK MODE ON # statement, when TIO receives a DC1 from the HP 3000, it sends a DC2 back. When the HP 3000 is ready to receive data in the Block mode, it again sends a DC1. TIO then activates the ON TRIGGER # interrupt.

When Block mode is enabled, the output data is not echoed back to the HP 250.

Successful execution of a REQUEST statement (to the HP 3000) sets Block mode off.

Example:

```
REQUEST Hp3000
BLOCK MODE ON #Hp3000
ON TRIGGER #Hp3000,5 GOSUB File_trans
 .
 .
BLOCK MODE OFF #Hp3000
```

One-Character Output

The HP 250 can send certain one-character output transmissions to the HP 3000 when the computer ready condition has not been established. For example, many HP 3000 utility programs respond to CTRL (Y) by terminating the current operation. The SEND statement sends the transmission.

SEND # device address , character code

The character code is a numeric expression which represents the character. ASCII character codes are listed in Appendix A of this manual. The character code for CTRL (Y) is 25.

If the state of the port has been enabled for output, the output state is terminated after the character code is sent.

Break Output

The program can break data transfer to the HP 3000 with the SEND BREAK # statement.

SEND BREAK # device address

If the state of the port was enabled for output, the output state is terminated.

Examples:

```
SEND BREAK #Hp3000

IF Flag=1 THEN SEND BREAK #15
```

BASIC Statements Used With TIO

Four BASIC statements are used frequently with TIO. The statements are described fully in the BASIC Programming Manual. They are briefly described here for your convenience.

REQUEST/RELEASE

Before any ON condition statement is executed, the program must have exclusive access of the required device.

Successful execution of the REQUEST statement gives the program exclusive use of a device. When the program is finished with the device, the RELEASE statement is used.

Successful execution of a REQUEST statement addressing a terminal causes TIO to implicitly execute the OFF INPUT # and ECHO ON # conditioning statements. For an HP 3000, TIO executes a BLOCK MODE OFF #.

REQUEST device address [, wait option]

If the wait option is omitted and the device is already reserved by another program, an error 131 results.

If the wait option is present, its initial value specifies whether or not the program is to be delayed if the device is already reserved. The wait option must be a variable.

Wait = 0 the requesting program may be delayed.
Wait ≠ 0 the requesting program may not be delayed.

If the initial value is 0, then TIO resets the value to indicate the results of the request attempt.

Wait = 0 exclusive access granted.
Wait = 1 device already reserved.

Exclusive access is relinquished by the RELEASE statement.

RELEASE device address

DISABLE/ENABLE

The DISABLE statement inhibits all interrupts (including ON KEY # interrupts); interrupts are still recorded. When the ENABLE statement is given, interrupts are serviced according to their priority. If two interrupts have the same priority, they are serviced according to their port number (port 15, then 14, 13, etc.).

DISABLE

ENABLE

The CURKEY Function

CURKEY is a function which returns a number indicating the source of an ON condition interrupt.

CURKEY numeric variable

The values CURKEY returns are shown in the following table:

Value	Condition
0	No interrupts have occurred
1-24	Softkeys 1-24
25-27	Port 1, device address 11
28-30	Port 2, device address 12
31-33	Port 3, device address 13
34-36	Port 4, device address 14
37-39	Port 5, device address 15

Three values are allocated for each port. An ON INPUT # or ON OUTPUT # interrupt returns the first value (25 for port 1, 28 for port 2, etc.). The second value is returned for an ON BREAK # interrupt. ON CONNECT # and ON DISCONNECT # interrupts cause the third value to be returned.

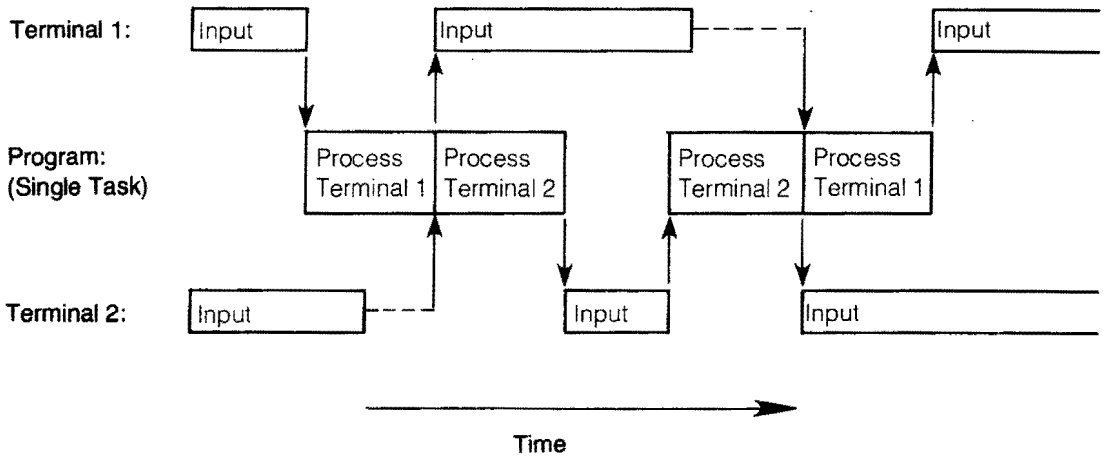
CHAPTER 3
Programming with TIO

Programming Overview

TIO/250 application programs can control remote terminals which are dedicated to the application. Since the terminal is not a remote console, the user is not confused by system messages or error codes. The application program can be tailored to the terminal user. Passwords and security features imbedded in the programs can control access to sensitive information.

The ON-condition statements overlap I/O and processing. For example, instead of waiting for a terminal to respond to a prompt, the program does other processing until a carriage return is received from the terminal. Depending upon relative priorities, the program can either accept the input immediately or finish the current processing before accepting the input. If the input is not received before processing is finished, the program uses the WAIT statement to explicitly wait for the input.

Overlapping I/O and processing of other tasks is particularly useful in applications where several terminals are serviced by a single program. For example:



Programming Tips

Input/Output States

The three types of I/O ports: terminal, computer, and printer, can be set in either an input or an output state. The printer is obviously an output device, thus the port is always in the output state. The computer port is always ready for input from the remote computer. When the remote computer sends a DC1 to the HP 250, the port will accept output, but it is still ready to accept input. Similarly when receiving input, one-character output (such as CTRL Y), or a break can be output.

A terminal port can be in only one state at a time. The program specifies which one. The state of a terminal at power-up or after a CTRL HALT is pressed is output. To enable input from a terminal, the ON INPUT # statement must be executed.

The change from output state to input state is never automatic. To re-enable output, successful execution of an OFF INPUT # or REQUEST statement is required. The output state is automatically enabled after an AREAD$ function and after a break is detected.

Branching Statements

The action and consequences of each of the branching statements should be understood well.

GOTO – When the interrupt condition is satisfied, execution branches to the line specified in the GOTO statement.

The execution priority is not changed.

```
10   REQUEST 11          Execution priority = 0
20   PRINTER IS 11
30   PRINT "ENTER YOUR NAME"
40   ON INPUT #11 GOTO 60
50   WAIT
60   A$=AREAD$(11)       Execution priority = 0
70    .
80    .
```

The GOTO statement is not recognized in other (subprogram) environments.

```
10   REQUEST 11
20   ON INPUT #11,5 GOTO 40
30   CALL X
40   A$=AREAD$(11)
 .
 .
100  SUB X      }
110  WAIT       }   The interrupt can never occur in this environ-
120  SUBEXIT    }   ment (X) because the branching statement is
                    not defined.
```

There is no "return" with a GOTO statement.

```
10    REQUEST 11
20    ON INPUT #11,2 GOTO X
 .
 .
400   WAIT
410 X:    A$=AREAD$(11)
```

If the interrupt occurs between line 20 and line 400, the program will branch out of some processing. It is impossible to return to complete the processing because the system does not remember where it was when the interrupt occurred.

GOSUB – Program execution branches to the subroutine specified when the interrupt condition is satisfied. When a RETURN is encountered, execution continues at the point the interrupt occurred.

The execution priority is raised to the value given in the ON-condition statement.

```
10    REQUEST 11                    Execution priority = 0
20    ON INPUT #11,3 GOSUB X
30    FOR I=1 TO 1E99
40     DISP I;
50    NEXT I
60    END
100 X:   A$=AREAD$(11)              Execution priority = 3
 .
 .
150   RETURN
```

The GOSUB statement is not recognized in other (subprogram) environments. This is the same as the GOTO statement.

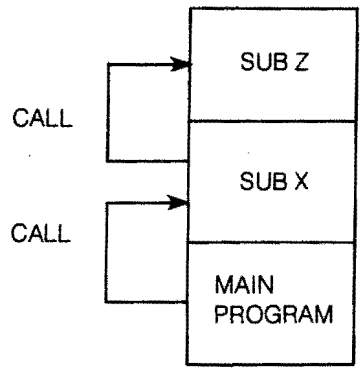
Execution resumes at the point of interruption after the subroutine has completed execution. In the example program above, if the interrupt occurred when line 40 was executing, execution would resume at line 50 after the subroutine was done.

CALL – Program execution in the current environment is suspended when the interrupt condition is satisfied. A new environment is created and remains in effect until a SUBEXIT or SUBEND is encountered or another CALL statement is executed.

The execution priority is raised to the value given in the ON-condition statement. This is the same as the GOSUB statement.

The CALL statement is recognized in all successive environments including the one containing the statement.

```
10    REQUEST 11
20    CALL X
 .
 .
 .
100   SUB X
110   ON INPUT #11 CALL Y
120   CALL Z
 .
 .
 .
200   SUB Z
 .
 .
 .
```



This program is a good example of how NOT to program with ON-condition statements. When subprogram X is called, the ON INPUT # statement is executed. The interrupt is defined in subprogram X and in subprogram Z. When both X and Z are exited, however, the interrupt is no longer defined.

NO BRANCHING STATEMENT – Informs the system of the programmer's intent to re-establish an interrupt condition in a previous environment. This serves to turn on the input or output interrupt. It is useful when switching environments or when changing a port's input/output state.

```
10    REQUEST 12
20    ON INPUT #12 CALL X
30    WAIT

100   SUB X
110   A$=AREAD$(12)

170   ON INPUT #12
180   SUBEXIT
```

Using ON OUTPUT #, OFF OUTPUT # or OFF INPUT # clears the branching statement of the ON INPUT # statement which was executed in the same environment. Therefore, an ON INPUT # with no branching statement will subsequently be ineffective.

```
10    REQUEST 12
20    ON INPUT #12 CALL X
30    CALL Y

100   SUB X
110   OFF INPUT #12
120   PRINTER IS 12
130   PRINT "..."

180   ON INPUT #12
190   SUBEXIT
```

The OFF INPUT #12 and ON INPUT #12 statements are executed in a different environment than the initial ON INPUT #12 statement.

Programming Approaches

Once TIO statements and the concepts are understood, you are ready to begin programming. The most useful program approaches are introduced next.

Straight Line Approach

The following program communicates with one terminal. It demonstrates the ease of programming for one remote device. Later, the same program will be expanded for multiple terminals.

```
10    DIM A$[254],B$[254]
20    Port=11
30    REQUEST Port
40    PRINTER IS Port,WIDTH(-1)
50    PRINT "Please enter your name:";
60    ON INPUT #Port GOTO In1
70    WAIT
80  In1:  A$=AREAD$(Port)
90    PRINT "What's your street address"&A$&"?"
100   ON INPUT #Port GOTO In2
110   WAIT
120 In2: B$=AREAD$(Port)
  .
  .
  .
```

Input data from the remote terminal; equivalent to LINPUT A$ directed to the main console.

Modular Approach

The modular approach to programming is useful when input from the terminal will determine which task is to be done.

In the following example, the program accepts a command from the terminal and the FNInterp function determines the task (X1, X2, ...) to be done.

```
10    DIM Commd$[254]
20    Port=11
30    REQUEST Port
40    PRINTER IS Port,WIDTH(-1)  !This is the default width for all TIO
devices.
50    PRINT "Please enter a command";LIN(1);":";
60    ON INPUT #Port GOSUB Service
70    WAIT
80    END
100 Service: Commd$=AREAD$(Port)
110   ON FNInterp(Commd$)+1 GOTO 120,140,160
120   PRINT "ERROR: COMMAND NOT RECOGNIZED."
130   GOTO 200
140   CALL X1(Commd$)
150   GOTO 200
160   CALL X2(Commd$)
170   GOTO 200
  .
200   PRINT ";";
210   ON INPUT #Port
220   RETURN
```

Array Addressing Mode

Expanding a program from accepting input from one terminal to accepting input from several terminals can be accomplished with little difficulty if the initial program was designed properly.

In the following example, the initial program is shown in the straight line approach example:

```
10    OPTION BASE 1
20    DIM A$(5)[254],B$(5)[254]
30    DISABLE
40    FOR Port=11 TO 15
50      REQUEST Port
60      PRINTER IS Port,WIDTH(-1)
70      PRINT "Please enter your name:";
80      ON INPUT #Port GOTO In1
90    NEXT Port
100   ENABLE
110   WAIT
200 In1: DISABLE
210   Port=(CURKEY-25)/3+11   !CALCULATE PORT NUMBER
220   A$(Port-10)=AREAD$(Port)
230   PRINTER IS Port,WIDTH(-1)
240   PRINT "What's your street address "&A$&"?"
250   ON INPUT #Port GOTO In2
260   ENABLE
270   WAIT
300 In2: DISABLE
310   Port=(CURKEY-25)/3+11
320   B$(Port-10)=AREAD$(Port)
```

The DISABLE and ENABLE statements are used to protect critical sections of code from GOTO interrupts.

This program can communicate with five terminals because the system always knows where to go when it gets an input line from a terminal. Having the system keep track of program state flow in this manner is called an **Implicit State Machine**. This is further discussed later in this chapter.

Executive Mode

In the following example the Service subroutine is used as an input/output traffic manager. It is referred to as an **Executive Routine.**

```
5     OPTION BASE 1
10    DIM A$[254],B$[254],Input$[254]
20    DIM Buff$(5)[750]
25    State=1
30       DISABLE
40       FOR Port=11 TO 15
50         PACK USING P1;Buff$(Port-10)
60         REQUEST Port
70         PRINTER IS Port
75         PRINT "Please enter your name:";
80         ON INPUT #Port GOSUB Service
90       NEXT Port
100      ENABLE
110      WAIT
120 P1:    PACKFMT State,A$,B$
130   !
200 Service:    Port=(CURKEY-25)/3+11
210      Input$=AREAD$(Port)
220      UNPACK USING P1;Buff$(Port-10)
230      PRINTER IS Port
240      GOSUB X
250      PACK USING P1;Buff$(Port-10)
260      ON INPUT #Port
270      RETURN
280   !
300 X:    ON State GOTO S1,S2,S3
310 S1:    A$=Input$
320        PRINT "What's your street address "&A$&"?"
330        State=2
340        RETURN
350   !
360 S2:    B$=Input$
```

This program demonstrates how PACK and UNPACK can be used to swap variables into and out of a common area. This program also illustrates how to use the **Explicit State Machine** approach.

Structured Programming

When the application program addresses one remote device with one task, you may only need to add a few statements to the current non-TIO program to drive that device. For example, assume a report is written at the end of the working day. Instead of outputting the report to the local (default) printer, the report is now to go to a remote printer. If this is the only remote I/O function the program performs, the only modifications needed are changing the device address in the PRINTER IS statements in the appropriate places.

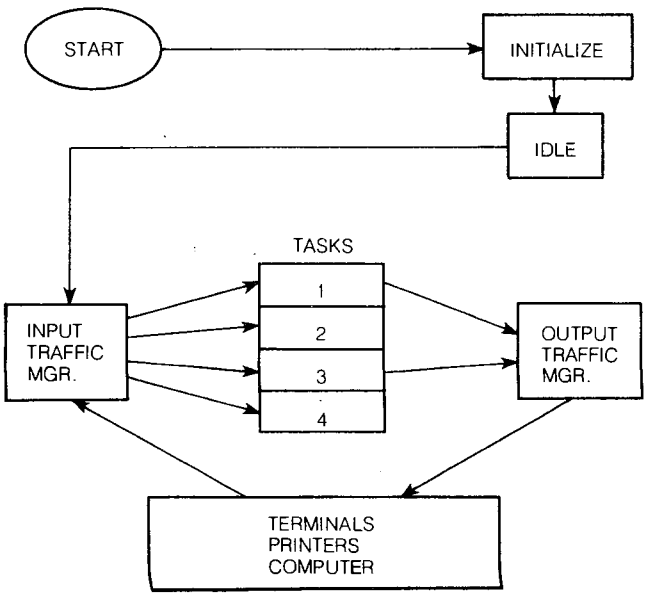
The creation of a more complex TIO application program, however, usually consists of many tasks which may be executing concurrently. In the TIO environment, each remote device may be associated with a task which controls the processing and input/output associated with that device. The primary problem is to prevent one task from interfering with the others. If interaction is desired among tasks, however, you must ensure that it happens efficiently.

The multi-tasking environment is more complex than that of sequential programing for one task. The problems demand a highly organized and structured approach. Without such an approach, you may have a program containing persistent (but unrepeatable, under debugging conditions) interference problems among the tasks.

Structured programming is concerned with improving the programming process through better program organization. Structured programming techniques, such as constructive use of subroutines and subprograms, ensure that a program is understandable, easily modified and documented, and easier to debug.

Basic Structural Flow

The following diagram can be used for many TIO applications.



INITIALIZE – The logic flow begins with the devices being initialized. This may include logging onto the HP 3000, testing if a printer is connected, or sending prompts to the terminals.

WAIT – Once the initialization is complete, the program waits for input from one or more of the devices.

INPUT TRAFFIC MGR – Input goes to the Input Traffic Manager routine which initiates a task.

TASKS – Each task is processed in the order of its priority.

OUTPUT TRAFFIC MGR – If output is required, the Output Traffic Manager initiates and completes it to the appropriate device.

TERMINALS, PRINTERS, COMPUTER – The program waits in an idle state until the next input from the devices or until one of the traffic managers begins the next task.

Structured Programming Diagram

Example Program

The program used to demonstrate the Executive Mode of programming was designed according to the preceding structural flow diagram.


```
5     OPTION BASE 1
10    DIM A$[254],B$[254],Input$[254]
20    DIM Buff$(5)[750]
25    State=1
30       DISABLE
40       FOR Port=11 TO 15
50         PACK USING P1;Buff$(Port-10)
60         REQUEST Port
70         PRINTER IS Port
75         PRINT "Please enter your name:";
80         ON INPUT #Port GOSUB Service
90       NEXT Port
100      ENABLE
110      WAIT
120 P1:    PACKFMT State,A$,B$
130   !
200 Service:    Port=(CURKEY-25)/3+11
210      Input$=AREAD$(Port)
220      UNPACK USING P1;Buff$(Port-10)
230      PRINTER IS Port
240      GOSUB X
250      PACK USING P1;Buff$(Port-10)
260      ON INPUT #Port
270      RETURN
280   !
300 X:    ON State GOTO S1,S2,S3
310 S1:    A$=Input$
320        PRINT "What's your street address "&A$&"?"
330        State=2
340        RETURN
350   !
360 S2:    B$=Input$
  .
  .
  .
```

Transaction Driven Applications

The design techniques described here are suitable for the creation of application programs driven by external events, such as remote terminals controlled by users. The application program and each user exchange data in an interactive fashion. The program usually supplies prompts to facilitate communication. The user might supply commands consisting of keywords and perhaps parameters to direct the program into specific operating modes and input data when requested by the current operating mode. In response to user commands (or perhaps by default), the program may display CRT forms to facilitate data entry and may generate and transmit reports either directly to the user's remote terminal or to some other output device. Finally, in response to user commands which cannot be fulfilled, the program generates messages which give the user the proper course of action.

A transaction consists of a logically complete interchange of prompts, commands, processing, input data and output reports. A transaction may be as simple as typing in a single command, in which case the transaction is just the action performed by that command. Transactions should be kept as simple as possible, otherwise the operational requirements (user training, program reliability, etc.) become extremely demanding.

Transactions may be catagorized as follows:

- **Security and overhead operations** such as user sign on and sign off.
- **Data retrieval operations** accessing a data base or a normal file in read-only mode. The objective of the data retrieval may be either quick "on-line" access to information or a printed report.
- **Batch data-entry operations** resulting in an intermediate or transaction file which is not the final end product of the application. The transaction file is later used as input in batch mode to a program which creates or updates the final end product (usually a data base).
- **Interactive data-entry operations** in which the data base is updated immediately, making the updated information available to other users as soon as the transaction is complete. If this technique is chosen, the parallel maintenance of a transaction file, as in batch data entry, should be strongly considered to allow backup and recovery from errors.

Generally, an application program offering interactive data entry transactions must ensure that multiple users and their associated tasks are protected from one another. In the worst case, the designer may have to prevent any other access to the data base while an update transaction is in progress. This includes data retrieval access in read-only mode since reporting of partially updated data may be unacceptable to the application.

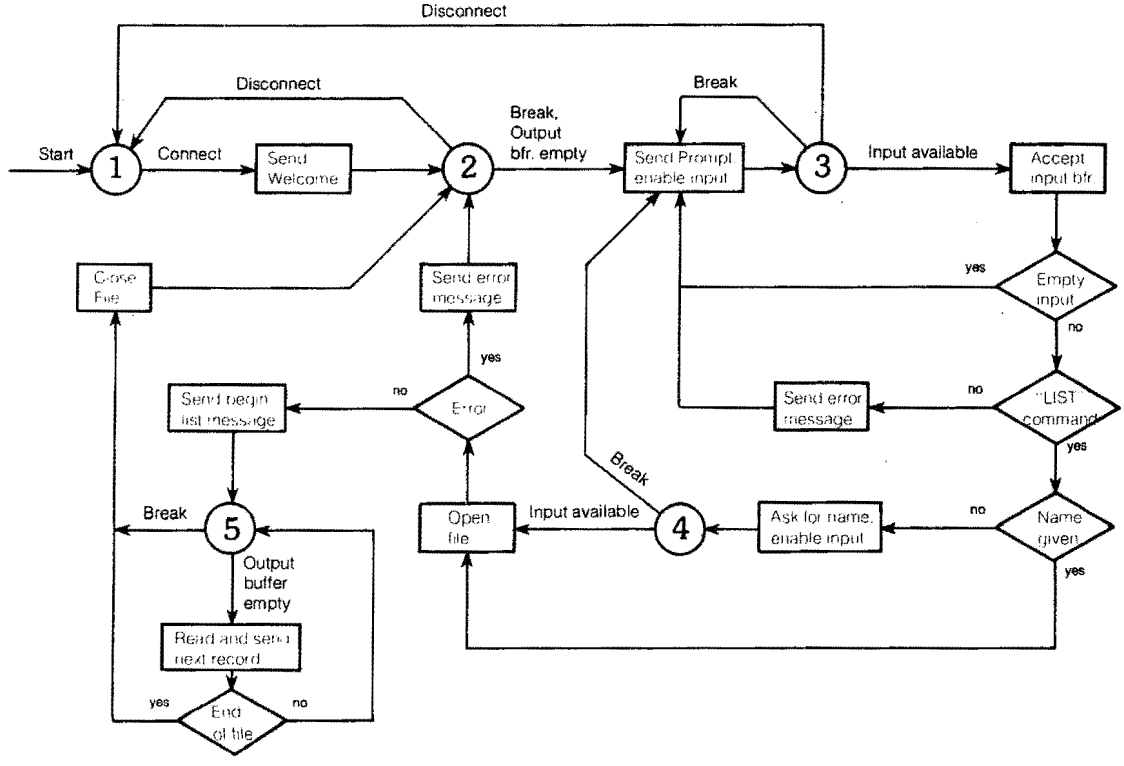
State Machine Model

The **State Machine Model** is a conceptual framework for designing a TIO application package. It is used to keep track of where you are in the program, and how you got there. For example, in the array addressing mode example, the variable Port is used to inform the system of which terminal gave input and where the input is to be stored.

In the executive mode example, the variable State is given a value when a routine has completed processing. When input is received from a terminal, State is used to determine the next task to perform. This method is called the **Explicit State Machine Approach** because the variable is explicitly assigned a value.

State Transition Diagrams

The designer applies the state machine model to his problem by creating a state transition diagram which specifies all possible states of a remote device, all acceptable external events for each state, the transition (that is, the next state) for each external event, and a brief description of the processing taking place during each transition. An example diagram is shown next.



State Transition Diagram

Example: File List Utility

This program outputs a file listing to the requesting terminal. It uses the example state transition diagram shown on the preceding page.

The program is initiated at the HP 250 main console and accepts a list of port numbers; the program attempts to attach each specified port. The program sends each connected terminal an identifying welcome message and the prompt character ">". The program accepts one command from the remote terminal: "LIST" (typed in either upper or lower case); the name of the desired file may follow the command name (separated by at least one blank). If the file name is not given in the "LIST" command, the remote terminal user is asked to supply the file name. The program then opens the file (assumed to consist of ASCII string data) and outputs the file to the user's terminal. The user may terminate the file listing at any time by pressing the terminal's BREAK key. When the end of the file is detected, the program informs the user, sends the prompt character, and awaits the next command. The single-line function, FNT, defined in line 60, returns the port number (11 thru 15) of the currently interrupting terminal. The function is recommended for all TIO programs.

```
10    !  HP 250 FILE LIST UTILITY FOR REMOTE TERMINALS
20    !
60    DEF FNT=INT((CURKEY-25)/3)+11                          State = 1.
70    INTEGER Port(1:5),I,W,Port
80    DIM C$[255],Bell$[1],Esc$[1],Home$[2],Clear$[2]
90    Bell$=CHR$(7)
100   Esc$=CHR$(27)
110   Home$=Esc$&"H"
120   Clear$=Esc$&"J"
130       DISP "HP 250 FILE LIST UTILITY FOR REMOTE TERMINALS"
140       DISP "Enter port numbers (each 1..5):  ,  ,  ,  ,  ";
150       CURSOR (XPOS-10)
160       INPUT "";Port(*)
170       DISABLE
180       FOR I=1 TO 5
190           IF Port(I) THEN GOSUB Startport
200       NEXT I
210       ON HALT GOTO Exit
220       ENABLE
230       WAIT
240   !
250 Startport:   Port=Port(I)+10
260       IF (Port>10) AND (Port<16) THEN 300
270          DISP "PORT";Port(I);"IS OUT OF RANGE"
280          Port(I)=0
290          RETURN
300       W=0
310       REQUEST Port,W
320       IF NOT W THEN 360
330          DISP "CAN'T GET EXCLUSIVE ACCESS TO PORT";Port(I)
340          Port(I)=0
350          RETURN
360       ON CONNECT #Port GOTO C1
370       RETURN
```

INITIALIZE

Ask main console for port # of terminals.

Check that port # is valid.

Request exclusive access.

```
380  !
390 Exit:   DISABLE
400       FOR I=1 TO 5
410          IF NOT Port(I) THEN RELEASE Port(I)+10
420       NEXT I
430       ENABLE
440       DISP "END OF PROGRAM"
450       STOP
460  !
470 C1:    DISABLE
480       PRINTER IS FNT
490       PRINT Home$&Clear$&"HP 250 FILE LISTER UTILITY"
500  !
510 C2:    ON DISCONNECT #FNT GOTO D1
520       ON OUTPUT #FNT GOTO O1
530       ON BREAK #FNT GOTO O1
540       ENABLE
550       WAIT
560  !
570 D1:    DISABLE
580       OFF OUTPUT #FNT
590       OFF BREAK #FNT
600       ON CONNECT #FNT GOTO C1
610       ENABLE
620       WAIT
630  !
640 O1:    DISABLE
650       PRINTER IS FNT
660  !
670 O2:    PRINT ")";
680       ON INPUT #FNT GOTO I1
690       ENABLE
700       WAIT
710  !
720 I1:    DISABLE
730       PRINTER IS FNT
740       C$=TRIM$(AREAD$(FNT))
750       IF NOT LEN(C$) THEN O2
760       IF POS(UPC$(C$)&" ","LIST ")=1 THEN List1
770       PRINT "Sorry, command not recognized."
780       GOTO O2
790  !
800 List1:   DISABLE
810       IF LEN(C$) THEN List3
820       PRINT  "Enter name of file to list: ";
830       ON INPUT #FNT GOTO List2
840       ENABLE
850       WAIT
860  !
870 List2:    DISABLE
880       PRINTER IS FNT
890       C$=TRIM$(AREAD$(FNT))
900       IF NOT LEN(C$) THEN O2
910  !
920 List3:    ON ERROR GOTO List7
930       ASSIGN #FNT-10 TO C$
940       BUFFER #FNT-10
950       ON END #FNT-10 GOTO List5
960       OFF ERROR
970       PRINT "Begin listing of file: ";C$;LIN(1)
980       ON OUTPUT #FNT GOTO List4
990       ON BREAK #FNT GOTO List6
1000      ON DISCONNECT #FNT GOTO List8
1010      ENABLE
1020      WAIT
```

INITIALIZE

Exit routine called when HALT is pressed.

Send welcome. State = 2.

End of file read, or break key pressed. State = 2.

ON DISCONNECT returns here. State = 1.

Send prompt.

State = 3.

Accept input.

Request file name. State = 4.

Receive file name.

State = 5.

```
1030 !
1040 List4:   DISABLE                          ┐
1050      PRINTER IS FNT                       │
1060      ON ERROR GOTO List7                  │
1070      READ #FNT-10;C$                      ├ Send record.
1080      OFF ERROR                            │
1090      PRINT C$                             │
1100      ENABLE                               │
1110      WAIT                                 ┘
1120 !
1130 List5:   ON ERROR GOTO 1150               ┐
1140      ASSIGN #FNT-10 TO *                  │
1150      OFF ERROR                            ├ Send end of file message.
1160      PRINT LIN(1);"END OF FILE"           │
1170      GOTO C2                              ┘
1180 !
1190 List6:   DISABLE                          ┐
1200      PRINTER IS FNT                       │
1210      ON ERROR GOTO 1230                   │
1220      ASSIGN #FNT-10 TO *                  ├ Break key was pressed.
1230      OFF ERROR                            │
1240      PRINT LIN(1);Bell$&"FILE LIST TERMINATED BY BREAK"
1250      GOTO C2                              ┘
1260 !
1270 List7:  ON ERROR GOTO 1290                ┐
1280      ASSIGN #FNT-10 TO *                  ├ Close file and print message.
1290      OFF ERROR                            │
1300      PRINT Bell$&"ERROR IN ACCESSING FILE "&C$
1310      GOTO C2                              ┘
1320 !
1330 List8:   DISABLE                          ┐
1340      ON ERROR GOTO 1360                   ├ Close file and go to disconnected routine.
1350      ASSIGN #FNT-10 TO *                  │
1360      OFF ERROR                            │
1370      GOTO D1                              ┘
```

This program implements one transaction type (the listing of a named file to the terminal). This is a data retrieval transaction. The program supports five remote terminals as readily as one, with no interference among the terminals (other than possible delays waiting for access to the same disc device). Except for initialization and termination, the program utilizes no variables other than the string buffer C$. Even this buffer is shared by all attached terminals. This is an extreme example of the implicit state marker technique; all state variable information associated with each terminal is maintained by the operating system and is invisible to the program. The program requires only five states to implement the specifications.

Interaction With Main Console Keyboard

A TIO application program may freely utilize the main console CRT and the 24 programmable keys. However, it is important to note that the system console is not a remote terminal. If the application program needs two terminals for input, you need two remote terminals, not one remote terminal and the system console. Therefore, special consideration is needed to set the interrupt priorities for the programmable keys relative to the priorities of the external (TIO-supplied) interrupts in accordance with the requirements of the application. The main console CRT serves as an application status monitor, showing the current activity of each remote terminal to the HP 250 operator.

If the application uses the main console keyboard as an input device, relative priorities must be decided. The program may be coded to provide for uninterruptable main keyboard input (invoked, say, by pressing a softkey). In this case, processing remote terminal interrupts is deferred until the main console operator presses ⇧.

Alternatively, the program may be coded to provide interruptable main keyboard input. Remote terminal interrupts continue to be processed during the input state. While an interrupt routine is in progress, the system removes the cursor from the main console CRT; keyboard input will not be processed and pressing any main console key will result only in a beep. When the interrupt routine exits, the cursor is restored and the input operation continues with no loss of previously accepted input. Note that the program must utilize GOSUB or CALL as the ON-condition branching statements to use this technique.