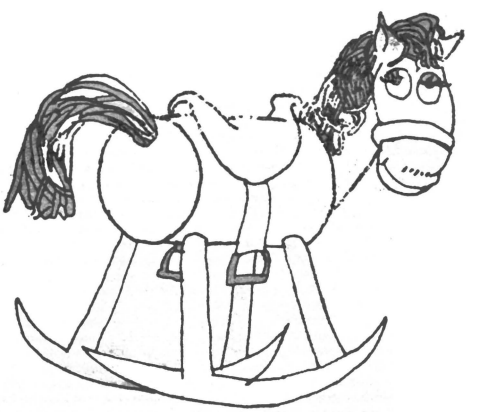
MISOSYS

MR. ED



Mister ED - A PRO-NTO Editor Application Pack

Mister ED - A PRO-NTO Editor Application Pack

Table of Contents

Mister ED is published by MISOSYS, Inc.

Note: Mister ED requires the PRO-NTO window and application manager.

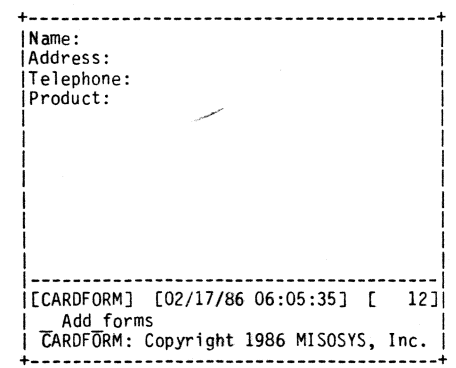
MISOSYS,Inc.
P.O. Box 239
Sterling, Virginia 22170-0239
703-450-4181

CARDFORM Application

The CARDFORM application is an adjunct to the CARD filer and notepad application. Although CARD was designed to provide a free-format text space of up to 480 characters of textual data, some peopl have been using it as a small file base storing fixed-format data (i.e. that which can be entered into a "form". Thus, we have had a few requests for a facility which can be used to populate a CARD file with a designated "form". CARDFORM provides such a capability. On the other hand, CARDFORM does not embellish CARD with any new facilities such as automatically skipping over the "fields" designated by the "form".

It is extremely important that you do NOT invoke CARDFORM while you are actively in the CARD application. If you are currently operating CARD and wish to add a few more records using the fixed format, you must exit CARD and then invoke CARDFORM. Likewise, you probably should not invoke CARD from CARDFORM. Although the specific reasons for these restraints don't really need to be discussed, you may be interested in knowing that the limitation is based on the problems which develop when a file is opened for reading and writing simultaneously by more than one application.

When you invoke the CARDFORM application, it will request PRO-NTO to open a window. A sample CARDFORM screen appears as follows:



When the window is opened, CARDFORM will search all disk drives for a file named CARD/DAT. When a CARD/DAT file is found, it will be accessed. CARDFORM will then scan through the data file until it locates a record with the ID field of "CARDFORM".If a window cannot be opened, a short beep will sound from your computer's internal speaker and CARD will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time. That's a rare instance, especially since you probably just exited CARD. If no file can be found identified as CARD/DAT, you will be notified of the

error and CARDFORM will terminate. If a record with the ID of "CARDFORM" cannot be found, you will be notified of that condition by the display of the message,

CARDFORM mask not found!

This essentially means that you have to exit CARDFORM and invoke CARD to prepare such a form (see Creating a CARDFORM record). Otherwise, the record which contains the "CARDFORM" form will be displayed and the CARDFORM command which is available. When CARDFORM is waiting for a command entry, the cursor which is at the beginning of the commands list will be blinking [this cursor is shown as an underline "_" in the preceding illustration].

There is only a single command which permits you to populate the CARD/DAT file with the displayed form. The command is invoked by depressing the letter key noted by the first letter of the command word (the letter "A" which appears capitalized). This may be entered in either upper or lower case.

Creating a CARD form record.

It is easy to create a form to be used by CARDFORM. Just invoke CARD, then "ADD" a record. Using the edit facility, compose your form in the text window of the added card. When you have completed and saved your edits, just invoke the "ID" command and identify the record as "CARDFORM". That's all there is to it.

ADD a form record

When you wish to add one or more records to your card file and also initialize the added record(s) with the cardform, specify this action by depressing the <A> key. You will be asked how many records you wish added by the prompt,

Add how many?

You may enter up to a three-digit decimal number. Terminate your entry with the <ENTER> key. After you have made your entry, CARDFORM will automatically add that many records and fill each record with the form. The DATE and TIME fields are automatically handled by CARDFORM which fills all added records with the current date and time. The ID field of the added records is left blank. If you decide that you do not want to add any records, just depress <BREAK>. CARDFORM will abort the addition and then await another command.

Please note that there is no way to go back to an existing record and merge in the form. CARDFORM can only insert the form into new records which it adds to the data file.

DED Application

The Disk EDitor (DED) application will allow you to edit a diskette on any DOS supported disk drive on a sector basis with full-screen editing. A sector is considered to be a 256-byte disk sector. The advantage of full screen editing is that you can make as many changes to a work copy of a sector without actually affecting the diskette until such time as you "save" your changes. DOS supported hard drives can be edited as easily as floppy diskettes.

Editing display screen

At all times during the editing of a sector, the work buffer for the selected sector will be displayed per the following screen layout:

```
  0123456789ABCDEF          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
================================================================================
| .*'>a.!.%..>|... | <00> | 00 2A 27 3E 61 EF 21 00 25 06 05 3E 7C EF C9 D5 |
| ..>|....>|..!.$> | <10> | 06 03 3E 7C EF E1 06 0A 3E 7C EF C9 21 00 24 3E |
| |........a..{... | <20> | 7C EF C9 01 00 00 18 F7 FE 61 D8 FE 7B D0 E6 DF |
| ..CARD/DAT..CARD | <30> | C9 1E 43 41 52 44 2F 44 41 54 03 1E 43 41 52 44 |
| FORM mask not fo | <40> | 46 4F 52 4D 20 6D 61 73 6B 20 6E 6F 74 20 66 6F |
| und...Add how ma | <50> | 75 6E 64 2E 03 1E 41 64 64 20 68 6F 77 20 6D 61 |
| ny? .[        ]  | <60> | 6E 79 3F 20 03 5B 20 20 20 20 20 20 20 20 5D 20 |
|  [                | <70> | 20 5B 20 20 20 20 20 20 20 20 20 20 20 20 5D 20 |
|    ] [     ]      | <80> | 20 20 20 5D 20 20 5B 20 20 20 20 20 20 5D 20 20 |
| Add forms. CARDF | <90> | 41 64 64 5F 66 6F 72 6D 73 0A 20 43 41 52 44 46 |
| ORM: Copyright 1 | <A0> | 4F 52 4D 3A 20 43 6F 70 79 72 69 67 68 74 20 31 |
| 986 MISOSYS, Inc | <B0> | 39 38 36 20 4D 49 53 4F 53 59 53 2C 20 49 6E 63 |
| ................ | <C0> | 2E 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ................ | <D0> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ................ | <E0> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ................ | <F0> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
================================================================================
Drive:  3  Cylinder: X'01' Sector: X'03' Byte: X'00' => X'11' =  17


Disk EDitor 1.1 - Copyright 1986 MISOSYS, Inc.
Command [; - @ A C D F G H I N P Q R S X Z]: _
```

The screen is divided into two major sections. The left hand section displays the sector's work buffer in ASCII. In this section, any sector byte value which is not a displayable ASCII character will be displayed as a period. The right hand section displays the sector's work buffer in hexadecimal; two hexadecimal digits describe the value of each byte in the sector. Each section is displayed in groups of sixteen bytes; thus, there are sixteen rows of sixteen bytes each for a total of 256 bytes per sector.

The actual byte location in the sector can be ascertained by the intersection of the high-order byte location with the low-order byte location. The

horizontal values across the top of the screen represent the low-order byte location of a sector byte. The vertical column of values enclosed in angle brackets (<00>, <10>, ...) represents the high-order location of a sector byte.

At all times, a pair of blinking cursors is positioned over one of the bytes in the work sector. One cursor is positioned over a byte in the ASCII section while the other is positioned at the corresponding byte location in the hexadecimal section. The status line appearing near the bottom of the screen displays information pertinent to the sector being displayed. This data is:

```
Drive:   3  Cylinder: X'01' Sector: X'03' Byte: X'00' => X'11' =  17
         a             bb            cc          dd       ee    fff
```

The "a" field contains the logical drive assignment of the disk drive currently being edited. The "bb" and "cc" fields contain the cylinder number and sector number of the disk sector currently in the work buffer. These numbers are relative to zero and are expressed as hexadecimal values. The "dd" field contains the location of the byte currently being pointed to by the buffer cursors. This field contains a hexadecimal value. The "ee" field contains the value of the byte at the buffer cursor. This value is displayed in hexadecimal. Finally, the "fff" field also contains the value of the byte at the buffer cursor but it is expressed in decimal.

The last line contains the command prompt. Enclosed within square brackets following the word "Command" is a brief list of the editing commands available to you. This terse list was designed only as a memory jogger; it is not considered to be a help screen. An underline cursor will be blinking at the end of this line to indicate an editing command entry is requested.

Invoking DED

DED can be invoked like any other application supplied with PRO-NTO. When you invoke the DED application, it will request PRO-NTO to open a 24 row by 80 column window. If the window cannot be opened, a short beep will sound from your computer's internal speaker and DED will terminate. When the window is opened, DED will prompt you for the logical drive number of the disk drive you wish to edit via the following prompt which will appear in the lower right hand portion of the screen:

Drive?

Respond with the logical drive number <0-7> of the disk you wish to edit. If you depress the <BREAK> key or the <EXPORT> key at this point, DED will terminate. If the disk designated according to your disk specification cannot be opened, the "Drive?" query will be redisplayed. When DED gains access to the designated disk drive, it will draw the display screen then await your command entry.

Summary of editing commands

The following command summary is presented for your use. These are the command keys and their functions for DED. Once you become familiar with the operation of DED, this section may be all you need to refer to from time to time to jog your memory. Here are DED's commands.

Key Entry	Function
<;>	Advance to the next sector
<->	Decrement to the previous sector
<@>	Move to the designated byte position
<A>	Enter ASCII modification mode
<C>	Find an ASCII string
<D>	Request a new DISK to edit
<F>	FIND a hexadecimal string
<G>	GO find the next search string match
<H>	Enter HEXADECIMAL modification mode
<I>	INSERT a null and push down
<N>	Move to the NEXT cylinder
<P>	Move to the PREVIOUS cylinder
<Q>	QUASH the current byte and pull up
<R>	REPOSITION to a designated sector
<S>	SAVE the work buffer to the file
<X>	EXIT the Disk EDitor
<Z>	ZAP to the end of the sector
<LEFT ARROW>	Move the cursor one position LEFT
<RIGHT ARROW>	Move the cursor one position RIGHT
<DOWN ARROW>	Move the cursor one position DOWN
<UP ARROW>	Move the cursor one position UP
<SHIFT LEFT>	Move the cursor to the beginning of the row
<SHIFT RIGHT>	Move the cursor to the end of the row
<SHIFT DOWN>	Move the cursor to the <FF> position
<SHIFT UP>	Move the cursor to the <00> position
<CLEAR LEFT>	Import text from previous screen

Cursor positioning manipulations

The <LEFT ARROW> moves the cursor one position to the left for each depression (or repeated key, if held down). When the cursor is positioned over the first character of a 16-byte row, a <LEFT ARROW> request will move the cursor to the last character position of the previous 16-byte row. When the cursor is positioned over the first byte of a sector (the <00> position), a <LEFT ARROW> will move the cursor to appear over the last byte of the sector.

The <RIGHT ARROW> request will move the cursor one position to the right. When the cursor is positioned over the last character position of a 16-byte row, a <RIGHT ARROW> request will move the cursor to the first column of the succeeding 16-byte row. When the cursor is positioned over the last

byte of a sector (the <FF> location), a <RIGHT ARROW> will move the cursor to appear over the first byte of the sector.

The <DOWN ARROW> request will move the cursor to the succeeding 16-byte row at the same column position. When the cursor is positioned in the last 16-byte row of the sector, a <DOWN ARROW> will move the cursor to appear in the same column of the first 16-byte row of the sector.

The <UP ARROW> request will move the cursor to the preceeding row at the same column position. When the cursor is positioned in the first 16-byte row of the sector, an <UP ARROW> will move the cursor to appear in the same column of the last 16-byte row of the sector.

The <SHIFT LEFT ARROW> request will move the cursor to the first position of the current row. The <SHIFT RIGHT ARROW> request will move the cursor to the last position of the current row. You can position the cursor to the first byte of the sector (the <00> position) by a <SHIFT UP ARROW> request. The <SHIFT DOWN ARROW> will position the cursor to appear over the last byte of the sector (the <FF> position).

Finally, the <@> request will enable you to move directly to any of the 256 positions of the displayed work sector. After invoking <@>, the "dd" status field will be blanked. You will then need to enter two hexadecimal digits which specify the desired byte location.

Sector positioning commands [<;> <-> <N> <P> <R>]

DED provides a handful of commands to select a particular sector of the disk for editing. If you have made any modifications to the currently displayed work sector which you have not saved to the disk, when you invoke one of these sector positioning commands, those changes will NOT be applied. Thus, it is easy to cancel any and all changes to the work sector prior to saving the sector to disk by repositioning to a different sector.

The <;> command advances one sector to display the sector which follows the current sector. If the currently displayed sector is the last sector of the cylinder being edited, DED will automatically advance to the first sector of the next cylinder. If the currently displayed sector is the last sector of the disk being edited, the <;> command will be ignored. The <;> is easily remembered as the lower case "+". This is also the same command key as is used by the system's DEBUG facility for incrementing the displayed page.

The <-> command decrements one sector to display the sector which precedes the current sector. If the currently displayed sector is the first sector of the cylinder being edited, DED will automatically decrement to the last sector of the previous cylinder. If the currently displayed sector is the first sector of the file being edited, the <-> command will be ignored. The <-> command key is the same command key as is used by the system's DEBUG facility for decrementing the displayed page.

The <N> command advances to the NEXT cylinder of the disk and displays the same numbered sector as is currently displayed. The cursor remains in the same byte position. Conversely, the <P> command decrements to the PREVIOUS cylinder of the disk and displays the same numbered sector as is currently displayed. As with the <N> command, the cursor remains in the same byte position after a <P> command.

You can select a particular cylinder and sector of the file for editing by using the <R> command. After you invoke this command, the "bb" field will be blanked and you will need to enter the number of the cylinder you wish to edit. This entry will be a 2-digit hexadecimal value. If you depress <ENTER> without making any entry, it will default to the first cylinder, <00>. After making the cylinder entry, the "cc" field will be blanked and you will need to enter the number of the sector on the previously designated cylinder that you wish to edit. This entry must also be a 2-digit hexadecimal value. If you depress the <ENTER> key without making an entry, the first sector will be selected, <00>. Don't forget that cylinder and sector numbers are relative to zero. This means that the first sector is 0, the second is 1, and so forth.

Making changes to the work sector [<A> <H> <I> <Q> <Z>]

DED provides five commands which you can use to make changes to the currently dislayed work buffer. Remember, the disk itself does not get changed until you SAVE the work buffer to the disk via the <S> command. In this way, you are given every opportunity to cancel any changes you have made prior to saving the buffer to disk.

The <A> command places DED into ASCII modification mode. In this mode, an entry of any printable ASCII character will replace the character beneath the ASCII section cursor. The cursor will then be advanced as if a <RIGHT ARROW> had then been entered. All eight cursor positioning command keys will be accepted for cursor repositioning while you are in ASCII modification mode. Depression of the <BREAK> key will terminate ASCII modification mode. DED will then be waiting for another command.

The <H> command places DED into HEXADECIMAL modification mode. In this mode, DED will be expecting the entry of two hexadecimal digits which will replace the byte value beneath the hexadecimal section cursor. Any entry which is not a hexadecimal digit [0-9, a-f, A-F] will be ignored; however, all eight cursor positioning command keys will be accepted. Depression of the <BREAK> key will terminate hexadecimal modification mode. DED will then be waiting for another command.

The <I> command will first push all the sector bytes starting with the byte value under the buffer cursor down by one byte position. This causes the byte currently in the <FF> position to be dropped. Then a byte value of <00> will be INSERTED into the location under the cursor.

The <Q> command will QUASH the byte currently under the buffer cursor by pulling up all the subsequent bytes by one position. Then a byte value of

<00> will be INSERTED into the <FF> location of the sector.

The <Z> command is used to ZAP all bytes of the sector, starting at the location of the buffer cursor, with a designated value. DED will prompt you for this value via the query,

Zap?

The value must be entered as two hexadecimal digits followed by an <ENTER>. If the entry is invalid, the command will be ignored; otherwise, the work buffer will be modified according to the buffer cursor location and the entered value.

Finding strings of bytes [<C> <F> <G>]

DED provides two SEARCH commands to scan the disk being edited for a specified string of characters. You specify the search by invoking either the <C> command or the <F> command. The <C> is used when you wish to enter the search string as ASCII characters whereas the <F> is used when you wish to enter the search string in pairs of hexadecimal digits. In either case, DED then prompts you for the search string with the query message,

String?

You can enter up to either 16 hexadecimal digits (8 2-digit pairs) or 8 ASCII characters to be used for the search string. Leave no spaces between hexadecimal digits in the case of the <F> command response. Terminate your search string with an <ENTER> (the <ENTER> character code is not included as one of the 16 hexadecimal digits or 8 characters).

If you enter a character for the <F> command query which is not a valid hexadecimal digit [0-9, a-f, A-F], you will see the following error message displayed;

Bad digit!

and the FIND request will be ignored. Otherwise, DED will then look for the string starting with the first character immediately following the buffer cursor. If the cursor is positioned at the <FF> location, the search will commence at the <00> position of the subsequent sector. DED will continue to scan to the end of the disk or until a match is found. The matching of alphabetic characters is case sensitive which means that characters entered in upper case must be found in upper case and characters entered in lower case must be found in lower case. If the search string cannot be found, the message,

String not found!

will be displayed. At this point, the cursor location remains unchanged. If, on the other hand, a matching string of text is found in the file, the sector

containing that string will be displayed. The cursor will be repositioned to the first byte of the matching string and the status information will be updated.

A way to find each occurrence of a search string is with the GO command, <G>. Each depression of <G> will find the next occurrence of the search string until no more matching strings can be located. At this point, the "String not found!" message as noted above will be displayed.

Saving the work sector to the disk file [<S>]

If you wish to save the current contents of the work sector to the disk being edited, you may do so by depressing <S>. DED will first ensure that you actually had intended to save the work buffer by prompting you with the query,

Save?

If you wish to proceed with the save operation, just depress the <ENTER> key. Any other entry except <EXPORT> will cancel the pending request. You can terminate DED without saving the work buffer and enter the export mode of PRO-NTO by depressing <CLEAR RIGHT ARROW>. The result will be the same as if you invoked <EXPORT> after requesting an exit from DED (via the <X> command). Thus, please refer to the discussion in the section on exiting from DED for the behaviour after such an <EXPORT> request.

After you save the work buffer (or cancel the request), DED is expecting the entry of another command.

Changing to a new DISK

The <D> command allows you to select another disk for editing. When you invoke the <D> command, you will be prompted for the name of the disk via the query,

Drive?

At this point, the operation of DED will be identical to the operation as if you had just entered DED. Respond with the logical drive number of the disk you wish to edit. If you depress the <BREAK> key or the <EXPORT> key at this point, DED will terminate. If the disk designated according to your disk specification cannot be accessed, the "Drive?" query will be redisplayed. When DED gains access to the designated disk drive, it will draw the display screen then await your command entry.

Exiting from DED [<X>]

When you have completed your edits and wish to exit DED, simply depress the <X> command. DED will first ensure that you actually had intended to terminate the application by prompting you with the query,

Exit?

If you want DED to terminate, just depress the <ENTER> key. Any other entry except <EXPORT> will cancel the pending request. If you wish to export the screen (or a portion of it) back to the interrupted program, depress <EXPORT> which is the <CLEAR RIGHT ARROW> key combination. DED will then enter the export mode. Don't forget that if you abort the export mode by depressing the <BREAK> key, PRO-NTO will return to the previously interrupted program as if export had not been invoked.

There are two ways of controlling what PRO-NTO does at the end of each exported line depending on how you mark the closure of the rectangle. If you close the rectangle via the <ENTER> key, a carriage return will be added to the "input" at the end of each marked line. This carriage return will be appended regardless of whether the marked rectangle is one or more lines. If the rectangle is closed by the depression of <SHIFT ENTER>, then the line is input from the beginning mark to the ending mark in a continuous stream; no carriage returns are added by PRO-NTO.

The rectangle is defined by the two points making up its northwest to southeast diagonal. These two points may be marked in the following manner:

1. Position the cursor to the upper left corner of the rectangle which will contain the information. The four arrow keys, <LEFT>, <RIGHT>, <UP>, and <DOWN>, will move the cursor around the screen.

2. Depress <CONTROL><B> to mark the beginning of the rectangle block. The character under the cursor will be replaced on the screen with a left square bracket which indicates the marked position. Don't worry about the bracket displayed; the correct character will be provided as input.

3. Position the cursor to the lower right corner of this rectangle again using the four arrow keys. This position may be on the same display row as the beginning mark.

4. Depress the <ENTER> or <SHIFT ENTER> key to mark the end of the marked block. This now defines the rectangle. The export will commence. The export function may be aborted anytime prior to marking the end of the rectangular block simply by depressing the <BREAK> key.

DOLOAD Application

The DOLOAD application will allow you to load a previously saved video screen display from a disk file into the video screen. Such a file may have been generated as 24 80-byte strings, each terminated by a carriage return, by the DOSAVE application. Why would you want to do such a thing? Well, the best reason would be to directly gain access to a video screen file which was previously saved by either the DOSAVE or VED applications. After access is easily achieved, the image or a portion of the image could be exported back to the application or program interrupted when DOLOAD was invoked.

When you invoke the DOLOAD application, it will request PRO-NTO to open a window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and DOLOAD will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time. That's a rare instance. When the window is opened, DOLOAD will prompt you to enter a file specification via the prompt,

Filespec?

The screen image that was saved in the disk file will be copied to the video screen. DOLOAD will then automatically terminate and enter the EXPORT mode of PRO-NTO. Thus, you then can conveniently mark a rectangular area of the display for export back to the interrupted program. Don't forget that if you abort the export mode by depressing the <BREAK> key, PRO-NTO will return to the previously interrupted program as if export had not been invoked.

There are two ways of controlling what PRO-NTO does at the end of each exported line depending on how you mark the closure of the rectangle. If you close the rectangle via the <ENTER> key, a carriage return will be added to the "input" at the end of each marked line. This carriage return will be appended regardless of whether the marked rectangle is one or more lines. If the rectangle is closed by the depression or <SHIFT ENTER>, then the line is input from the beginning mark to the ending mark in a continuous stream; no carriage returns are added by PRO-NTO.

The rectangle is defined by the two points making up its northwest to southeast diagonal. These two points may be marked in the following manner:

1. Position the cursor to the upper left corner of the rectangle which will contain the information. The four arrow keys, <LEFT>, <RIGHT>, <UP>, and <DOWN>, will move the cursor around the screen.

2. Depress <CONTROL><B> to mark the beginning of the rectangle block. The character under the cursor will be replaced on the screen with a left square bracket which indicates the marked position. Don't worry about the bracket displayed; the correct character will be provided as input.

3. Position the cursor to the lower right corner of this rectangle again using the four arrow keys. This position may be on the same display row as the beginning mark.

4. Depress the <ENTER> or <SHIFT ENTER> key to mark the end of the marked block. This now defines the rectangle. The export will commence. The export function may be aborted anytime prior to marking the end of the rectangular block simply by depressing the <BREAK> key.

FED Application

The File EDitor (FED) application will allow you to edit any disk file on a record basis with full-screen editing. A record is considered to be a 256-byte disk sector. The advantage of full screen editing is that you can make as many changes to a work copy of a record without actually affecting the disk file until such time as you "save" your changes.

Editing display screen

At all times during the editing of a record, the work buffer for the selected record will be displayed per the following screen layout:

```
 0123456789ABCDEF          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
================================================================================
| PRONTOFile EDito | <00> | 50 52 4F 4E 54 4F 46 69 6C 65 20 45 44 69 74 6F |
| r ............. | <10> | 72 20 03 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| .(.(........... | <20> | 1D 28 1C 28 00 00 00 00 00 00 00 00 00 00 00 00 |
| ............... | <30> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ............... | <40> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ............... | <50> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ............... | <60> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ............... | <70> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ............... | <80> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ............... | <90> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ............... | <A0> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ............... | <B0> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| ............... | <C0> | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
| 02/11/8615:05:48 | <D0> | 30 32 2F 31 31 2F 38 36 31 35 3A 30 35 3A 34 38 |
| Copyright (c) 19 | <E0> | 43 6F 70 79 72 69 67 68 74 20 28 63 29 20 31 39 |
| 86 MISOSYS, Inc. | <F0> | 38 36 20 4D 49 53 4F 53 59 53 2C 20 49 6E 63 2E |
================================================================================
File: FED/APP:7                  Sector:     0  Byte: X'00' => X'50' =  80

File EDitor 1.1 - Copyright 1986 MISOSYS, Inc.
Command [; - @ A B C E F G H I N Q R S X Z]: _
```

The screen is divided into two major sections. The left hand section displays the record's work buffer in ASCII. In this section, any record byte value which is not a displayable ASCII character will be displayed as a period. The right hand section displays the record's work buffer in hexadecimal; two hexadecimal digits describe the value of each byte in the record. Each section is displayed in groups of sixteen bytes; thus, there are sixteen rows of sixteen bytes each for a total of 256 bytes per record.

The actual byte location in the record can be ascertained by the intersection of the high-order byte location with the low-order byte location. The horizontal values across the top of the screen represent the low-order byte location of a record byte. The vertical column of values enclosed in angle

brackets (<00>, <10>, ...) represents the high-order location of a record byte.

At all times, a pair of blinking cursors is positioned over one of the bytes in the work record. One cursor is positioned over a byte in the ASCII section while the other is positioned at the corresponding byte location in the hexadecimal section. The status line appearing near the bottom of the screen displays information pertinent to the record being displayed. This data is:

```
File: FED/APP:7                Sector:     0  Byte: X'00' => X'50' =  80
      aaaaa.....................          bbbbb          cc        dd     eee
```

The "aaaaa" field contains the specification of the file currently being edited. The "bbbbb" field contains the number of the record expressed in decimal. This number is relative to zero for the first record. The "cc" field contains the location of the byte currently being pointed to by the buffer cursors. This field contains a hexadecimal value. The "dd" field contains the value of the byte at the buffer cursor. This value is displayed in hexadecimal. Finally, the "eee" field also contains the value of the byte at the buffer cursor but it is expressed in decimal.

The last line contains the command prompt. Enclosed within square brackets following the word "Command" is a brief list of the editing commands available to you. This terse list was designed only as a memory jogger; it is not considered to be a help screen. An underline cursor will be blinking at the end of this line to indicate an editing command entry is requested.

Invoking FED

FED can be invoked like any other application supplied with PRO-NTO. When you invoke the FED application, it will request PRO-NTO to open a 24 row by 80 column window. If the window cannot be opened, a short beep will sound from your computer's internal speaker and FED will terminate. When the window is opened, FED will prompt you for the name of the disk file you wish to edit via the following prompt which will appear in the lower right hand portion of the screen:

```
Filespec?
```

Respond with the file specification of the file you wish to edit. If you depress the <BREAK> key or the <EXPORT> key at this point, FED will terminate. If the file designated according to your file specification cannot be opened, the "Filespec?" query will be redisplayed. When FED gains access to the designated disk file, it will draw the display screen then await your command entry.

Summary of editing commands

The following command summary is presented for your use. These are the command keys and their functions for FED. Once you become familiar with the operation of FED, this section may be all you need to refer to from time to time to jog your memory. Here are FED's commands.

Key Entry	Function
<;>	Advance to the next record
<->	Decrement to the previous record
<@>	Move to the designated byte position
<A>	Enter ASCII modification mode
<B>	Move to the BEGINNING of the file
<C>	Find an ASCII string
<E>	Move to the END of the file
<F>	FIND a hexadecimal string
<G>	GO find the next search string match
<H>	Enter HEXADECIMAL modification mode
<I>	INSERT a null and push down
<N>	Request a NEW file to edit
<Q>	QUASH the current byte and pull up
<R>	Position to a designated RECORD
<S>	SAVE the work buffer to the file
<X>	EXIT the File EDitor
<Z>	ZAP to the end of the record
<LEFT ARROW>	Move the cursor one position LEFT
<RIGHT ARROW>	Move the cursor one position RIGHT
<DOWN ARROW>	Move the cursor one position DOWN
<UP ARROW>	Move the cursor one position UP
<SHIFT LEFT>	Move the cursor to the beginning of the row
<SHIFT RIGHT>	Move the cursor to the end of the row
<SHIFT DOWN>	Move the cursor to the <FF> position
<SHIFT UP>	Move the cursor to the <00> position
<CLEAR LEFT>	Import text from previous screen

Cursor positioning manipulations

The <LEFT ARROW> moves the cursor one position to the left for each depression (or repeated key, if held down). When the cursor is positioned over the first character of a 16-byte row, a <LEFT ARROW> request will move the cursor to the last character position of the previous 16-byte row. When the cursor is positioned over the first byte of a record (the <00> position), a <LEFT ARROW> will move the cursor to appear over the last byte of the record.

The <RIGHT ARROW> request will move the cursor one position to the right. When the cursor is positioned over the last character position of a 16-byte row, a <RIGHT ARROW> request will move the cursor to the first column of the succeeding 16-byte row. When the cursor is positioned over the last

byte of a record (the <FF> location), a <RIGHT ARROW> will move the cursor to appear over the first byte of the record.

The <DOWN ARROW> request will move the cursor to the succeeding 16-byte row at the same column position. When the cursor is positioned in the last 16-byte row of the record, a <DOWN ARROW> will move the cursor to appear in the same column of the first 16-byte row of the record.

The <UP ARROW> request will move the cursor to the preceeding row at the same column position. When the cursor is positioned in the first 16-byte row of the record, an <UP ARROW> will move the cursor to appear in the same column of the last 16-byte row of the record.

The <SHIFT LEFT ARROW> request will move the cursor to the first position of the current row. The <SHIFT RIGHT ARROW> request will move the cursor to the last position of the current row. You can position the cursor to the first byte of the record (the <00> position) by a <SHIFT UP ARROW> request. The <SHIFT DOWN ARROW> will position the cursor to appear over the last byte of the record (the <FF> position).

Finally, the <@> request will enable you to move directly to any of the 256 positions of the displayed work record. After invoking <@>, the "cc" status field will be blanked. You will then need to enter two hexadecimal digits which specify the desired byte location.

Record positioning commands [<;> <-> <B> <E> <R>]

FED provides a handful of commands to select a particular record of the file for editing. If you have made any modifications to the currently displayed work record which you have not saved to the disk file, when you invoke one of these record positioning commands, those changes will NOT be applied. Thus, it is easy to cancel any and all changes to the work record prior to saving the record to disk by repositioning to a different record.

The <;> command advances one record to display the record which follows the current record. If the currently displayed record is the last record of the file being edited, the <;> command will be ignored. The <;> is easily remembered as the lower case "+". This is also the same command key as is used by the system's DEBUG facility for incrementing the displayed page.

The <-> command decrements one record to display the record which precedes the current record. If the currently displayed record is the first record of the file being edited, the <-> command will be ignored. The <-> command key is the same command key as is used by the system's DEBUG facility for decrementing the displayed page.

The <B> command captures the first record of the file as the work buffer and displays that record. The cursor is positioned to the <00> position. Conversely, the <E> command captures the last record of the file as the work buffer and displays that record. In this case, the cursor is positioned to

the byte in that record which is the last byte of the file according to the information in the system's directory entry for that file.

You can select a particular record of the file for editing by using the <R> command. After you invoke this command, the "bbbbb" field will be blanked and you will need to enter the number of the record you wish to edit. This entry will be a DECIMAL value. If you depress <ENTER> without making any decimal entry, the first record will be selected which is the same as the <B> command. Don't forget that record numbers are relative to zero. This means that the first record is 0, the second is 1, and so forth.

Making changes to the work record [<A> <H> <I> <Q> <Z>]

FED provides five commands which you can use to make changes to the currently dislayed work buffer. Remember, the file itself does not get changed until you SAVE the work buffer to the file via the <S> command. In this way, you are given every opportunity to cancel any changes you have made prior to saving the buffer to disk.

The <A> command places FED into ASCII modification mode. In this mode, an entry of any printable ASCII character will replace the character beneath the ASCII section cursor. The cursor will then be advanced as if a <RIGHT ARROW> had then been entered. All eight cursor positioning command keys will be accepted for cursor repositioning while you are in ASCII modification mode. Depression of the <BREAK> key will terminate ASCII modification mode. FED will then be waiting for another command.

The <H> command places FED into HEXADECIMAL modification mode. In this mode, FED will be expecting the entry of two hexadecimal digits which will replace the byte value beneath the hexadecimal section cursor. Any entry which is not a hexadecimal digit [0-9, a-f, A-F] will be ignored; however, all eight cursor positioning command keys will be accepted. Depression of the <BREAK> key will terminate hexadecimal modification mode. FED will then be waiting for another command.

The <I> command will first push all the record bytes starting with the byte value under the buffer cursor down by one byte position. This causes the byte currently in the <FF> position to be dropped. Then a byte value of <00> will be INSERTED into the location under the cursor.

The <Q> command will QUASH the byte currently under the buffer cursor by pulling up all the subsequent bytes by one position. Then a byte value of <00> will be INSERTED into the <FF> location of the record.

The <Z> command is used to ZAP all bytes of the record, starting at the location of the buffer cursor, with a designated value. FED will prompt you for this value via the query,

Zap?

The value must be entered as two hexadecimal digits followed by an <ENTER>. If the entry is invalid, the command will be ignored; otherwise, the work buffer will be modified according to the buffer cursor location and the entered value.

Finding strings of bytes [<C> <F> <G>]

FED provides two SEARCH commands to scan the file being edited for a specified string of characters. You specify the search by invoking either the <C> command or the <F> command. The <C> is used when you wish to enter the search string as ASCII characters whereas the <F> is used when you wish to enter the search string in pairs of hexadecimal digits. In either case, FED then prompts you for the search string with the query message,

String?

You can enter up to either 16 hexadecimal digits (8 2-digit pairs) or 8 ASCII characters to be used for the search string. Leave no spaces between hexadecimal digits in the case of the <F> command response. Terminate your search string with an <ENTER> (the <ENTER> character code is not included as one of the 16 hexadecimal digits or 8 characters).

If you enter a character for the <F> command query which is not a valid hexadecimal digit [0-9, a-f, A-F], you will see the following error message displayed;

Bad digit!

and the FIND request will be ignored. Otherwise, FED will then look for the string starting with the first character immediately following the buffer cursor. If the cursor is positioned at the <FF> location, the search will commence at the <00> position of the subsequent record. The matching of alphabetic characters is case sensitive which means that characters entered in upper case must be found in upper case and characters entered in lower case must be found in lower case. If the search string cannot be found, the message,

String not found!

will be displayed. At this point, the cursor location remains unchanged. If, on the other hand, a matching string of text is found in the file, the record containing that string will be displayed. The cursor will be repositioned to the first byte of the matching string and the status information will be updated.

A way to find each occurrence of a search string is with the GO command, <G>. Each depression of <G> will find the next occurrence of the search string until no more matching strings can be located. At this point, the "String not found!" message as noted above will be displayed.

Saving the work record to the disk file [<S>]

If you wish to save the current contents of the work record to the disk file being edited, you may do so by depressing <S>. FED will first ensure that you actually had intended to save the work buffer by prompting you with the query,

Save?

If you wish to proceed with the save operation, just depress the <ENTER> key. Any other entry except <EXPORT> will cancel the pending request. You can terminate FED without saving the work buffer and enter the export mode of PRO-NTO by depressing <CLEAR RIGHT ARROW>. The result will be the same as if you invoked <EXPORT> after requesting an exit from FED (via the <X> command). Thus, please refer to the discussion in the section on exiting from FED for the behaviour after such an <EXPORT> request.

After you save the work buffer (or cancel the request), FED is expecting the entry of another command.

Changing to a NEW file

The <N> command allows you to select another file for editing. When you invoke the <N> command, you will be prompted for the name of the file via the query,

Filespec?

At this point, the operation of FED will be identical to the operation as if you had just entered FED. Respond with the file specification of the file you wish to edit. If you depress the <BREAK> key or the <EXPORT> key at this point, FED will terminate. If the file designated according to your file specification cannot be opened, the "Filespec?" query will be redisplayed. When FED gains access to the designated disk file, it will draw the display screen then await your command entry.

Exiting from FED [<X>]

When you have completed your edits and wish to exit FED, simply depress the <X> command. FED will first ensure that you actually had intended to terminate the application by prompting you with the query,

Exit?

If you want FED to terminate, just depress the <ENTER> key. Any other entry except <EXPORT> will cancel the pending request. If you wish to export the screen (or a portion of it) back to the interrupted program, depress <EXPORT> which is the <CLEAR RIGHT ARROW> key combination. FED will then enter the

export mode. Don't forget that if you abort the export mode by depressing the <BREAK> key, PRO-NTO will return to the previously interrupted program as if export had not been invoked.

There are two ways of controlling what PRO-NTO does at the end of each exported line depending on how you mark the closure of the rectangle. If you close the rectangle via the <ENTER> key, a carriage return will be added to the "input" at the end of each marked line. This carriage return will be appended regardless of whether the marked rectangle is one or more lines. If the rectangle is closed by the depression of <SHIFT ENTER>, then the line is input from the beginning mark to the ending mark in a continuous stream; no carriage returns are added by PRO-NTO.

The rectangle is defined by the two points making up its northwest to southeast diagonal. These two points may be marked in the following manner:

1. Position the cursor to the upper left corner of the rectangle which will contain the information. The four arrow keys, <LEFT>, <RIGHT>, <UP>, and <DOWN>, will move the cursor around the screen.

2. Depress <CONTROL><B> to mark the beginning of the rectangle block. The character under the cursor will be replaced on the screen with a left square bracket which indicates the marked position. Don't worry about the bracket displayed; the correct character will be provided as input.

3. Position the cursor to the lower right corner of this rectangle again using the four arrow keys. This position may be on the same display row as the beginning mark.

4. Depress the <ENTER> or <SHIFT ENTER> key to mark the end of the marked block. This now defines the rectangle. The export will commence. The export function may be aborted anytime prior to marking the end of the rectangular block simply by depressing the <BREAK> key.

MED Application

The Memory EDitor (MED) application will allow you to edit any random access memory (RAM) page on a page basis with full-screen editing. A page is considered to be a 256-byte block of RAM origined on an <xx00> basis in the address space of the computer (<xx00> is expressed in hexadecimal). The advantage of full screen editing is that you can make as many changes to a work copy of a page without actually affecting the memory until such time as you "save" your changes.

Editing display screen

At all times during the editing of a page, the work buffer for the selected page will be displayed for the following screen layout:

```
 0123456789ABCDEF          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
================================================================================
| ASM.).N....*.-.. | <00> | 41 53 4D D1 29 DD 4E 06 E5 09 EB 2A F9 2D E5 B7 |
| .B".-.../....... | <10> | ED 42 22 F9 2D E1 CD 8C 2F 03 EB D1 CD AD 2E 11 |
| A'....W../.q).F. | <20> | 41 27 DD 00 00 F5 57 CD 11 2F CD 71 29 DD 46 07 |
| *.)..*....S/..z/ | <30> | 2A BF 29 CD AF 2A 10 F8 C9 CD 53 2F C0 CD 7A 2F |
| ..(+.../.*.)..-. | <40> | D2 F3 28 2B E5 CD 8C 2F E5 2A D1 29 CD FE 2D F5 |
| *...W.W.W......_ | <50> | 2A 00 00 F5 57 F4 57 F5 57 00 00 8A 00 08 00 5F |
| .......).BDM...C/ | <60> | 00 00 00 00 07 D1 29 ED 42 44 4D 03 E1 CD 43 2F |
| ...O..x.......a. | <70> | F1 C1 E1 30 01 09 78 B1 C8 ED B8 BF C9 FE 61 D8 |
| .{.. ...7*.-../> | <80> | FE 7B D0 D6 20 C9 11 D0 37 2A F9 2D CD 89 2F 3E |
| .... .+..7.../.T | <90> | FF EB ED B1 20 11 2B 11 CF 37 E5 CD 8C 2F E1 54 |
| ]>.............. | <A0> | 5D 3E FE ED B9 00 00 00 00 00 00 00 00 00 00 00 |
| FED/ASM - 02/11/ | <B0> | 46 45 44 2F 41 53 4D 20 2D 20 30 32 2F 31 31 2F |
| 86.;*=*=*..OPTIO | <C0> | 38 36 0D 3B 2A 3D 2A 3D 2A 0D 09 4F 50 54 49 4F |
| N.CI.;*=*=*.*GET | <D0> | 4E 09 43 49 0D 3B 2A 3D 2A 3D 2A 0D 2A 47 45 54 |
|  SVCMAC.;*=*=*.C | <E0> | 20 53 56 43 4D 41 43 0D 3B 2A 3D 2A 3D 2A 0D 43 |
| MDPOS.EQU.23.SHL | <F0> | 4D 44 50 4F 53 09 45 51 55 09 32 33 2E 53 48 4C |
================================================================================
Banks: 31  Current Bank:   2 Page: X'80' Byte: X'00' => X'41' =  65

Memory EDitor 1.0 - Copyright 1986 MISOSYS, Inc.
Command [; - @ A B C D F G H I N P Q S X Z]: -
```

The screen is divided into two major sections. The left hand section displays the page's work buffer in ASCII. In this section, any page byte value which is not a displayable ASCII character will be displayed as a period. The right hand section displays the page's work buffer in hexadecimal; two hexadecimal digits describe the value of each byte in the page. Each section is displayed in groups of sixteen bytes; thus, there are sixteen rows of sixteen bytes each for a total of 256 bytes per page.

The actual byte location in the page can be ascertained by the intersection of the high-order byte location with the low-order byte location. The

horizontal values across the top of the screen represent the low-order byte location of a page byte. The vertical column of values enclosed in angle brackets (<00>, <10>, ...) represents the high-order location of a page byte.

At all times, a pair of blinking cursors is positioned over one of the bytes in the work buffer. One cursor is positioned over a byte in the ASCII section while the other is positioned at the corresponding byte location in the hexadecimal section. The status line appearing near the bottom of the screen displays information pertinent to the page being displayed. This data is:

```
Banks:  31  Current Bank:   2 Page: X'80' Byte: X'00' => X'41' =  65
        aa                  bb         cc          dd       ee    fff
```

The "aa" field contains the maximum number of memory banks accessible to the DOS and is shown in decimal [the "31" shown in the example reflects a Model 4 equipped with a one megabyte memory hardware option]. The "bb" field contains the currently selected memory bank and is also shown in decimal. The "cc" field contains the memory page currently in the work buffer. This number is relative to zero for the first page and is expressed in hexadecimal. The page contents of the memory bank specified by the "bb" field will be in the work buffer when this "cc" field is designated within the range <80> through <FF>. The "dd" field contains the location of the byte currently being pointed to by the buffer cursors. This field contains a hexadecimal value. The "ee" field contains the value of the byte at the buffer cursor. This value is displayed in hexadecimal. Finally, the "fff" field also contains the value of the byte at the buffer cursor but it is expressed in decimal.

The last line contains the command prompt. Enclosed within square brackets following the word "Command" is a brief list of the editing commands available to you. This terse list was designed only as a memory jogger; it is not considered to be a help screen. An underline cursor will be blinking at the end of this line to indicate an editing command entry is requested.

Invoking MED

MED can be invoked like any other application provided with PRO-NTO. When you invoke the MED application, it will request PRO-NTO to open a 24 row by 80 column window. If the window cannot be opened, a short beep will sound from your computer's internal speaker and MED will terminate. When the window is opened, MED will will draw the display screen defaulting to bank 0, page 0; then await your command entry.

Summary of editing commands

The following command summary is presented for your use. These are the command keys and their functions for MED. Once you become familiar with the operation of MED, this section may be all you need to refer to from time to time to jog your memory. Here are MED's commands.

Key Entry	Function
<;>	Advance to the next page
<->	Decrement to the previous page
<@>	Move to the designated byte position
<A>	Enter ASCII modification mode
<B>	Request a specific BANK of memory
<C>	Find an ASCII string
<D>	DISPLAY a specified page of memory
<F>	FIND a hexadecimal string
<G>	GO find the next search string match
<H>	Enter HEXADECIMAL modification mode
<I>	INSERT a null and push down
<N>	Advance to the NEXT bank of memory
<P>	Decrement to the PREVIOUS bank of memory
<Q>	QUASH the current byte and pull up
<S>	SAVE the work buffer to memory
<X>	EXIT the Memory EDitor
<Z>	ZAP to the end of the page
<LEFT ARROW>	Move the cursor one position LEFT
<RIGHT ARROW>	Move the cursor one position RIGHT
<DOWN ARROW>	Move the cursor one position DOWN
<UP ARROW>	Move the cursor one position UP
<SHIFT LEFT>	Move the cursor to the beginning of the row
<SHIFT RIGHT>	Move the cursor to the end of the row
<SHIFT DOWN>	Move the cursor to the <FF> position
<SHIFT UP>	Move the cursor to the <00> position
<CLEAR LEFT>	Import text from previous screen

Cursor positioning manipulations

The <LEFT ARROW> moves the cursor one position to the left for each depression (or repeated key, if held down). When the cursor is positioned over the first character of a 16-byte row, a <LEFT ARROW> request will move the cursor to the last character position of the previous 16-byte row. When the cursor is positioned over the first byte of a page (the <00> position), a <LEFT ARROW> will move the cursor to appear over the last byte of the page.

The <RIGHT ARROW> request will move the cursor one position to the right. When the cursor is positioned over the last character position of a 16-byte row, a <RIGHT ARROW> request will move the cursor to the first column of the succeeding 16-byte row. When the cursor is positioned over the last byte of a page (the <FF> location), a <RIGHT ARROW> will move the cursor to

appear over the first byte of the page.

The <DOWN ARROW> request will move the cursor to the succeeding 16-byte row at the same column position. When the cursor is positioned in the last 16-byte row of the page, a <DOWN ARROW> will move the cursor to appear in the same column of the first 16-byte row of the page.

The <UP ARROW> request will move the cursor to the preceeding row at the same column position. When the cursor is positioned in the first 16-byte row of the page, an <UP ARROW> will move the cursor to appear in the same column of the last 16-byte row of the page.

The <SHIFT LEFT ARROW> request will move the cursor to the first position of the current row. The <SHIFT RIGHT ARROW> request will move the cursor to the last position of the current row. You can position the cursor to the first byte of the page (the <00> position) by a <SHIFT UP ARROW> request. The <SHIFT DOWN ARROW> will position the cursor to appear over the last byte of the page (the <FF> position).

Finally, the <@> request will enable you to move directly to any of the 256 positions of the displayed work page. After invoking <@>, the "dd" status field will be blanked. You will then need to enter two hexadecimal digits which specify the desired byte location.

Page positioning commands [<;> <-> <B> <D> <N> <P>]

MED provides a handful of commands to select a particular page of the file for editing. If you have made any modifications to the currently displayed work page which you have not saved to memory, when you invoke one of these page positioning commands, those changes will NOT be applied. Thus, it is easy to cancel any and all changes to the work page prior to saving the page to memory by repositioning to a different page.

The <;> command advances one page to display the page which follows the current page. If the currently displayed page is the last page in the machine's address space (page <FF>), the <;> command will advance to page <00>. The <;> is easily remembered as the lower case "+". This is also the same command key as is used by the system's DEBUG facility for incrementing the displayed page.

The <-> command decrements one page to display the page which precedes the current page. If the currently displayed page is the first page of the address space (the <00> page), the <-> command will advance to page <FF>. The <-> command key is the same command key as is used by the system's DEBUG facility for decrementing the displayed page.

The <B> command is used to select another BANK of RAM for accessing pages in the <80> through <FF> range. When you invoke this command, the "bb" field will be blanked. You are expected to enter up to a 2-digit decimal bank number (relative to zero). Terminate your entry with the <ENTER> key. An

invalid bank selection designation will be ignored.

The <N> command advances the bank selection to the NEXT higher numbered bank. If the highest numbered bank is currently selected, this command is ignored. Conversely, the <P> command decrements the bank selection to the previous bank (the next lower numbered bank). If bank <0> had been selected, the command will be ignored.

In each of the bank selection commands, (<B>, <N>, and <P>), if the displayed page is currently in the <80> through <FF> range, the work buffer will be loaded with the contents of the memory page from the selected bank. The display will be redrawn with the contents of the buffer and the cursor position remains unchanged. If the currently displayed page is in the <00> through <7F> range, only the status line is updated to reflect the new bank selection.

You can DISPLAY a particular page of memory for editing by using the <D> command. After you invoke this command, the "cc" field will be blanked and you will need to enter the number of the page you wish to edit. This entry will be a HEXADECIMAL value. If you depress <ENTER> without making any entry, the first page will be selected (page <00>). Don't forget that page numbers are relative to zero. This means that the first page is 0, the second is 1, and so forth, with the last being <FF>.

Making changes to the work page [<A> <H> <I> <Q> <Z>]

MED provides five commands which you can use to make changes to the currently dislayed work buffer. Remember, memory itself does not get changed until you SAVE the work buffer to memory via the <S> command. In this way, you are given every opportunity to cancel any changes you have made prior to saving the buffer to memory.

The <A> command places MED into ASCII modification mode. In this mode, an entry of any printable ASCII character will replace the character beneath the ASCII section cursor. The cursor will then be advanced as if a <RIGHT ARROW> had then been entered. All eight cursor positioning command keys will be accepted for cursor repositioning while you are in ASCII modification mode. Depression of the <BREAK> key will terminate ASCII modification mode. MED will then be waiting for another command.

The <H> command places MED into HEXADECIMAL modification mode. In this mode, MED will be expecting the entry of two hexadecimal digits which will replace the byte value beneath the hexadecimal section cursor. Any entry which is not a hexadecimal digit [0-9, a-f, A-F] will be ignored; however, all eight cursor positioning command keys will be accepted. Depression of the <BREAK> key will terminate hexadecimal modification mode. MED will then be waiting for another command.

The <I> command will first push all the page bytes starting with the byte value under the buffer cursor down by one byte position. This causes the

byte currently in the <FF> position to be dropped. Then a byte value of <00> will be INSERTED into the location under the cursor.

The <Q> command will QUASH the byte currently under the buffer cursor by pulling up all the subsequent bytes by one position. Then a byte value of <00> will be INSERTED into the <FF> location of the page.

The <Z> command is used to ZAP all bytes of the page, starting at the location of the buffer cursor, with a designated value. MED will prompt you for this value via the query,

Zap?

The value must be entered as two hexadecimal digits followed by an <ENTER>. If the entry is invalid, the command will be ignored; otherwise, the work buffer will be modified according to the buffer cursor location and the entered value.

Finding strings of bytes [<C> <F> <G>]

MED provides two SEARCH commands to scan memory for a specified string of characters. You specify the search by invoking either the <C> command or the <F> command. The <C> is used when you wish to enter the search string as ASCII characters whereas the <F> is used when you wish to enter the search string in pairs of hexadecimal digits. In either case, MED then prompts you for the search string with the query message,

String?

You can enter up to either 16 hexadecimal digits (8 2-digit pairs) or 8 ASCII characters to be used for the search string. Leave no spaces between hexadecimal digits in the case of the <F> command response. Terminate your search string with an <ENTER> (the <ENTER> character code is not included as one of the 16 hexadecimal digits or 8 characters).

If you enter a character for the <F> command query which is not a valid hexadecimal digit [0-9, a-f, A-F], you will see the following error message displayed;

Bad digit!

and the FIND request will be ignored. Otherwise, MED will then look for the string starting with the first character immediately following the buffer cursor. If the cursor is positioned at the <FF> location, the search will commence at the <00> position of the subsequent page. When MED reaches the end of page <FF> during the search, it will automatically continue the scan starting with page <80> of the NEXT bank until it either reaches page <FF> of the highest bank or finds a matching string. The matching of alphabetic characters is case sensitive which means that characters entered in upper case must be found in upper case and characters entered in lower case must be

found in lower case. If the search string cannot be found, the message,

String not found!

will be displayed. At this point, the cursor location remains unchanged. If, on the other hand, a matching string of text is found in memory, the page containing that string will be displayed. The cursor will be repositioned to the first byte of the matching string and the status information will be updated.

A way to find each occurrence of a search string is with the GO command, <G>. Each depression of <G> will find the next occurrence of the search string until no more matching strings can be located. At this point, the "String not found!" message as noted above will be displayed.

Saving the work page to memory [<S>]

If you wish to save the current contents of the work page to memory, you may do so by depressing <S>. MED will first ensure that you actually had intended to save the work buffer by prompting you with the query,

Save?

If you wish to proceed with the save operation, just depress the <ENTER> key. Any other entry except <EXPORT> will cancel the pending request. You can terminate MED without saving the work buffer and enter the export mode of PRO-NTO by depressing <CLEAR RIGHT ARROW>. The result will be the same as if you invoked <EXPORT> after requesting an exit from MED (via the <X> command). Thus, please refer to the discussion in the section on exiting from MED for the behaviour after such an <EXPORT> request.

After you save the work buffer (or cancel the request), MED is expecting the entry of another command.

Exiting from MED [<X>]

When you have completed your edits and wish to exit MED, simply depress the <X> command. MED will first ensure that you actually had intended to terminate the application by prompting you with the query,

Exit?

If you want MED to terminate, just depress the <ENTER> key. Any other entry except <EXPORT> will cancel the pending request. If you wish to export the screen (or a portion of it) back to the interrupted program, depress <EXPORT> which is the <CLEAR RIGHT ARROW> key combination. MED will then enter the export mode. Don't forget that if you abort the export mode by depressing the <BREAK> key, PRO-NTO will return to the previously interrupted program as if export had not been invoked.

There are two ways of controlling what PRO-NTO does at the end of each exported line depending on how you mark the closure of the rectangle. If you close the rectangle via the <ENTER> key, a carriage return will be added to the "input" at the end of each marked line. This carriage return will be appended regardless of whether the marked rectangle is one or more lines. If the rectangle is closed by the depression of <SHIFT ENTER>, then the line is input from the beginning mark to the ending mark in a continuous stream; no carriage returns are added by PRO-NTO.

The rectangle is defined by the two points making up its northwest to southeast diagonal. These two points may be marked in the following manner:

1. Position the cursor to the upper left corner of the rectangle which will contain the information. The four arrow keys, <LEFT>, <RIGHT>, <UP>, and <DOWN>, will move the cursor around the screen.

2. Depress <CONTROL><B> to mark the beginning of the rectangle block. The character under the cursor will be replaced on the screen with a left square bracket which indicates the marked position. Don't worry about the bracket displayed; the correct character will be provided as input.

3. Position the cursor to the lower right corner of this rectangle again using the four arrow keys. This position may be on the same display row as the beginning mark.

4. Depress the <ENTER> or <SHIFT ENTER> key to mark the end of the marked block. This now defines the rectangle. The export will commence. The export function may be aborted anytime prior to marking the end of the rectangular block simply by depressing the <BREAK> key.

REGENBU Utility

The REGENBU/BAS program was written to be used as a maintenance tool for the BRINGUP/APP application supplied with PRO-NTO. BRINGUP itself provides facilities for removing unneeded entries from the data file; however, the action of removing an entry does not shrink the file. It merely deactivates the activity slot so it can be used for another ADD. Thus, the operation of BRING ensures that the BRINGUP/DAT file will always be as large as the largest number of active activities ever stored. This may not prove practical in some cases. The REGENBU utility can remedy the problem of a maintaining a large BRINGUP/DAT file which is storing few active activities.

The "REGENBU/BAS" program is written in BASIC. It is adapted from the BRINGUP/BAS program listed in the PRO-NTO manual. REGENBU/BAS operates by reading the BRINGUP/DAT file and creating a new file called ACTIVE/DAT which contains only active activity records from the old file. If you ever need to shrink your BRINGUP/DAT file after removing a number of deleted records, you can run this program from BASIC.

The steps to "regenerate" your BRINGUP/DAT file using REGENBU/BAS are as follows:

1. Invoke the program via **BASIC REGENBU/BAS**

2. After its operation, rename your existing BRINGUP/DAT file to some archival name such as "BU120685/DAT".

3. Rename the "ACTIVE/DAT" file to "BRINGUP/DAT".

After confirmation that the new data file is correct, you can archive the old file to some archive diskette and remove it from your working disk.

This page intentionally left blank

TED Application

The TED application is a full screen "quick" text editor with typical word-processing type features (four-directional cursor movement; two-directional scrolling; text insertion and overstrike; string search and replace; block copy, delete, and move; directional delete; large text buffer; etc); however, TED was not designed to be a full featured word processor. Because of the limited space available for running PRO-NTO applications, some compromises were made in the installation of features. Prompting and error messages were necessarily made brief to leave the maximum amount of space available for programming editing functions. In spite of our excellent programming efforts, you may not find every editing function you would like to see in a text editor; however, we have done our best in squeezing in as many functions as we could to make TED a useful text editor for the purpose to which TED was targeted. TED was designed for you to be able to rapidly enter a full-screen text editing environment while you are running some other program. We believe that we have met that goal. TED, of course, runs under PRO-NTO - the windowing application manager from MISOSYS!

TED incorporates some pretty sophisticated memory management techniques in providing you with a text buffer which can hold up to 30 thousand characters of text. However, TED cannot pull a rabbit out of a hat. In order to utilize TED, you must have one 32K external bank of RAM free for its use. TED will search for a free RAM bank out of the possible 31 banks available when your machine is equipped with a DOS supported memory add-on board. If you are using a 128-K machine, you will have a free memory bank after PRO-NTO is installed unless you are using it for a MemDISK or unless the program which you interrupt to invoke TED has already claimed the available bank. TED uses this external memory bank to swap the contents of the user address space (starting from 3000H). On exit from TED, it's text buffer is again swapped with the original contents of the user address space which was saved in the external memory. Thus, you can re-capture the contents of the text buffer after you have terminated TED by gaining access to the memory bank which TED had used. This is discussed in the section on text recovery with the OOPS application.

For its small size, TED boasts an impressive list of features:

- Editing window: 22 lines of 80 characters each; 30K text buffer.
- Text entry modes: overstrike mode, insert mode.
- Word wraps at end of line; not word bounce.
- Text delete modes: char, block, to eol, to bol, to top, to bottom.
- Cursor movement: left, right, up, down, top, bottom, bol, eol.
- String search (23 character string).
- String search and replace (23 character string).
- Scroll up, scroll down, page up, page down.
- Block modes: mark begin, mark end, copy, delete, and move.
- Load text from a disk file; Concatenate at current end of text.
- File text to disk; File extension default (/TXT).
- User definable command key set (via altering with FED).
- User definable overstrike/insert cursors (via altering with FED).

Summary of editing commands

The following command summary is presented for your use. These are the command keys and their functions as supplied by TED. If you choose to alter any of the command keystrokes, please make note of your changes. Once you become familiar with the operation of TED, this section may be all you need to refer to from time to time to jog your memory. Here are TED's commands.

Key Entry	Function
<^A>	Toggle overstrike/insert modes
<^B>	Specify BLOCK...
<^D>	Specify DELETE
<^F>	FILE the text buffer to disk
<^G>	GO find the next search string match
<^H>	Same as the <LEFT ARROW> key
<^I>	Same as the <RIGHT ARROW> key
<^J>	Same as the <DOWN ARROW> key
<^K>	Same as the <UP ARROW> key
<^L>	LOAD a text file into the buffer
<^M>	Same as the <ENTER> key
<^N>	Go to the NEXT video page
<^R>	REPLACE searched string with new string
<^S>	SEARCH for a string
<^U>	Go UP to the previous video page
<CLEAR SHIFT =>	EXIT the text editor
<LEFT ARROW>	Move the cursor one position left
<RIGHT ARROW>	Move the cursor one position right
<DOWN ARROW>	Move the cursor one position down
<UP ARROW>	Move the cursor one position up
<SHIFT LEFT>	Move the cursor to the beginning of the line
<SHIFT RIGHT>	Move the cursor to the end of the line
<SHIFT DOWN>	Move the cursor to the end of the text
<SHIFT UP>	Move the cursor to the beginning of the text
<CLEAR LEFT>	Import text from previous screen

Invoking TED

TED is invoked like any other application supplied with PRO-NTO. When you invoke the TED application, it will request PRO-NTO to open a full-screen (80x24) window. If a window cannot be opened, a short beep will sound from your computer's internal speaker and TED will terminate. This happens when you have exceeded the maximum number of windows that can be open at one time which is a rare instance. When the window is opened, the video screen will clear and TED will search for a free external memory bank. If one is not available, a short beep will sound from your computer's internal speaker and TED will terminate. You can differentiate between the two condi- tions which cause an immediate "beep" and termination by noting whether or not the screen cleared. Remember, if a window is not available, the screen will not clear; if a memory bank is not available, the screen will have been cleared.

Once TED successfully completes the first two steps, it will display a welcome message on the 24th line of the video screen. This display line will also be used for the display of status, prompting, and error messages. TED displays three different types of messages during its operation. Error messages are indicated by a terminating exclamation point, "!". Queries which need a response are indicated by a terminating question mark, "?". Informative messages use no special character for their termination. Thus, "Marker!" is an error, "String?" is a query, and "Block" is information.

The first twenty two lines of the video screen are used as a display window for the text file under edit. This text area is separated from the status display line by a horizontal line drawn with the underbar character.

Throughout this documentation, keystrokes are denoted by being enclosed in angle brackets. Most of TED's commands are entered with the control key (labeled <CTRL>). Some use the <CLEAR> key, instead. Throughout this documentation, the control key will be noted by the caret character, "^". Thus, for example, the "block" command will be denoted as <^B>.

Text entry modes

TED will accept only displayable ASCII characters in the range 20H through 7FH for text entry. You can ascertain which characters this represents by displaying the machine's character set with the CHARSET application which was provided with PRO-NTO. Any other character value will be interpreted as a command entry. If it matches a value in the command table, that command will be invoked; otherwise, the entry will be ignored.

TED operates in two text entry modes: overstrike and insert. The initial mode established when TED is first invoked is the "overstrike" mode. TED can be modified to initially enter the "insert" mode, if you so desire, by changing the value of a single byte in the application file itself. The method of accomplishing this is explained in the section covering alterations to TED.

While TED is in "overstrike" mode, it will use an underbar as the cursor character (character value 5FH as displayed by CHARSET). When you toggle to "insert" mode, the cursor is changed to a full graphics block (character value BFH). You toggle from one mode to the other via the <^A> command.

When TED is in "overstrike" mode, any acceptable text entry typed character is written over the character which appears under the blinking cursor. You can overstrike a newline character (i.e. <ENTER>, which is displayed as character value 84H - see CHARSET). You can also overstrike either a "begin" block marker (character value B7H) or an "end" block marker (character value B8H). You can be in overstrike mode when you come to the end of the text (or starting from an empty text buffer, for that matter) and still be able to enter text in this mode.

When you switch to "insert" mode, anytime you enter an acceptable text entry character, the entire text will be pushed down one position starting from the character under the cursor to make room for the inserted character. The video screen will be constantly updated as text is inserted.

The text entry mode is only changed via the <^A> command. Going into "delete" mode does not change the mode of text entry.

Internally, TED uses a NULL character (character value 00H) to indicate the terminating position of the text buffer. Throughout this documentation, the word "NULL" denotes this facility.

Exiting from TED

It is easy to exit TED and return to the program which was interrupted to invoke TED. The <CLEAR SHIFT => command tells TED you wish to exit. If the text buffer is empty, TED will immediately terminate. However, if there is any text in the buffer, you are provided an opportunity to retract your request. TED will display the prompt message,

Sure?

If you wish to exit TED, all you need do is depress the <ENTER> key and TED will terminate. If you wish to EXPORT any portion of the screen contents back to the previously interrupted program, depress PRO-NTO's EXPORT command, <CLEAR RIGHT ARROW>. This is the only point during TED's operation at which the EXPORT facility may be requested. Any other keystroke entered in response to the "Sure?" prompt will be interpreted as a retraction of the EXIT request.

If you inadvertantly exit TED and wish you hadn't because you forgot to save the text, you may be in luck. See the section on text recovery with the OOPS application.

Loading a text file

The <^L> command is used to load a text file into the text buffer area. When you depress <^L>, you will be prompted for the name of the file via the prompt message,

Filespec

If the file you wish to load has an extension of "/TXT", you do not have to enter the extension. If the extension is omited from your entry, "/TXT" will be automatically provided.

The LOAD command will not automatically clear any text remaining in the text buffer prior to the LOAD. This means that the new text from the disk file will be concatenated to the old text in the editing buffer. The new text

is not inserted at the cursor position but rather is appended to the end of the current text. If you wish to load the new file over the old text, simply invoke the command sequence, <CLEAR UP ARROW> followed by <^D> then <CLEAR DOWN ARROW>. This will delete the entire text buffer. Alternatively, you can invoke a <CLEAR DOWN ARROW> then follow it with a <^D> then <CLEAR UP ARROW>. Both sequences will delete all of the text in the buffer. You can then load in the new file.

If the file is too large to fit into the available text buffer, the error message,

No room

will be displayed and no text will be loaded. If any disk read error is encountered while reading the text file into the text buffer, the message,

I/O error!

will be displayed. The text which was loaded up to the point of encountering the error will be retained in the text buffer.

Entering text

Entering text is easy, you just type away. If you already have text in the buffer and wish to enter new text at the end, just move the cursor to the bottom (via the <CLEAR DOWN ARROW> key), then type in your text. If you wish to enter new text at some other point, just position the cursor, toggle to the "insert" mode, then type away. TED will stay in "insert" mode until explicitly toggled back to "overstrike" mode.

As you are entering text, any word which is too long to fit at the end of a video line will be split at the 80th column and continued onto the next line. These "long words" are not automatically bounced onto the subsequent line, as is the case with the typical "word processor".

TED also provides you with other facilities for editing text. You can perform a "cut and paste" operation by first marking a block of text and then moving it to where you want the marked block positioned. You can "search and replace" a find text string of up to 23 characters with a replacement text string of up to 23 characters. Sorry, but due to space restrictions, the search is case sensitive.

Don't forget, you can nearly always invoke another PRO-NTO application from TED. This means that you can use the EXPORT facility from the invoked application to pass data back to TED as text input. TED permits the use of PRO-NTO's IMPORT facility during text entry and during the input of search/replace strings and file specifications. Thus, you can directly IMPORT text from the interrupted program's screen into TED's text buffer. On the other hand, you could also interrupt a program and save the screen image into a disk file with DOSAVE (or VED) and then load that file into TED's text

buffer. A little ingenuity can work wonders.

Cursor positioning manipulations

In all cases concerning cursor manipulation, the last position of a line is interpreted according to the following priority:

1. the NULL which is used by TED to indicate the end of the text;
2. a new line character, the <ENTER>;
3. or the 80th character on the line.

This means that when TED needs to determine what is the last displayed character position on a line, it will first check for case 1. Failing that, it will check for case 2. Failing that, the last character position will be the 80th column, by default. TED provides you with many commands to position the cursor and/or display different portions of your text. The ARROW keys are the primary tools to move the cursor. These keys will be interpreted as cursor movement requests unless TED is in the DELETE or BLOCK modes.

The <LEFT ARROW> moves the cursor one position to the left for each depression (or repeated key, if held down). When the cursor is positioned over the first character of a line, a <LEFT ARROW> request will move the cursor to the last character position of the previous line.

The <RIGHT ARROW> request will move the cursor one position to the right. When the cursor is positioned over either a new line character or the 80th character position on the line, a <RIGHT ARROW> request will move the cursor to the first column of the succeeding line. TED will not advance the cursor past the terminating NULL.

The <DOWN ARROW> request will move the cursor to the succeeding line at the same column position. However, if that position would be beyond the last position of that line, the cursor will be repositioned to the last position of that line. If the line position of the cursor had been the last line of the text portion displayed on the video screen, the text will be scrolled up one row to make the succeeding line visible. TED will not advance the cursor past the terminating NULL.

The <UP ARROW> request will move the cursor to the preceeding line at the same column position. However, if that position would be beyond the last position of that line, the cursor will be repositioned to the last position of that line. If the line position of the cursor had been the first line of the text portion displayed on the video screen, the text will be scrolled down one row to make the preceeding line visible. Obviously, if the first line displayed was the first line of your text, the <UP ARROW> request to change the cursor position will be ignored.

The <SHIFT LEFT ARROW> request will move the cursor to the first position of the current line. The <SHIFT RIGHT ARROW> request will move the cursor to the last position of the current line. You can position the cursor to the first position of the text buffer by a <SHIFT UP ARROW> request. If that text position had not been on the video display, the screen will be refreshed so that the top of the text will be displayed starting from the top of the video display. Finally, the <SHIFT DOWN ARROW> will refresh the screen so that the line which contains the terminating NULL will be displayed at the top of the video screen and the cursor will be positioned over the NULL.

The page up, <^U>, command will refresh the video screen so that the new first displayed line is twenty one lines previous to the current first displayed line. That also means the new last displayed line was the first line displayed prior to the page up request. This all assumes that there are at least twenty one lines preceeding the top line. If there are fewer than twenty one lines, the result will be as if you had invoked a TOP command, <SHIFT UP ARROW>.

The page next, <^N>, request will refresh the video screen so that the new first line displayed is the last line of the current displayed text. If the video display has fewer than 22 lines of text displayed, the page next request will be ignored.

After either a page up or page next request, the new cursor position will be the home position of the screen (i.e. row 0, column 0).

Text deletion

TED provides five forms of text deletion in addition to the block deletion discussed later. To delete the single character which appears under the cursor, invoke the delete command via <^D>. This action will get rid of the character and all text which succeeded that character will be pulled back one position. The <^D> command also puts you into DELETE mode which is made apparent by the display of the word,

Delete

in the status line. The DELETE mode is active for only the next keyboard entry. There are only four subcommands associated with the DELETE mode: delete to beginning of line (bol), delete to end of line (eol), delete to top, and delete to bottom. These subcommands are specified by the cursor movement keys associated with cursor positioning. To refresh your memory, use the following table:

Deletion desired	Command sequence
delete to bol	<^D> then <SHIFT LEFT ARROW>
delete to eol	<^D> then <SHIFT RIGHT ARROW>
delete to top	<^D> then <SHIFT UP ARROW>
delete to bottom	<^D> then <SHIFT DOWN ARROW>

After the <^D> is requested, the character now under the cursor is the character which was to the right of the deleted character. Since in the case of delete to bol and delete to top, you are deleting text which is in front of the cursor, you really don't want to delete the character which is under the cursor after the <^D>. Well, you don't have to worry about that because those two subcommands properly backup one position before continuing the deletion.

By the way, TED considers all four of these subcommand deletions as severe; thus, it will issue the "Sure?" query and expect the entry of <ENTER> in order to carry out the deletion.

Block operations

The BLOCK command, <^B>, has five subcommands: Begin, End, Copy, Delete, and Move. These subcommands are specified by entering the first letter of the subcommand word (<B>, <E>, <C>, <D>, or <M>). The entry may be in either upper or lower case. Note that these subcommands are NOT control key combinations but normal alphabetic single-key entries. When you invoke the BLOCK command, the word,

Block

will be displayed in the status line.

A block deletion request is considered by TED to be severe enough to warrant double checking your request before going ahead and performing the deletion. Thus, the operation of deleting a block was designed to be a BLOCK-Delete function sequence (<^B>, <D>) rather than a DELETE-BLOCK function sequence (<^D>, <^B>) to avoid the single character deletion which always occurs with the DELETE command.

Anytime you need to deal with a block; say to copy it, move it, or delete it, you have to first mark it. The beginning and ending positions of a block are marked by first positioning the cursor over the first character of the block and then entering the two command sequence, <^B> followed by <B>. This is followed up be positioning the cursor over the character immediately following the last character of the block and then entering the two command sequence, <^B> followed by <E>. The beginning position will be indicated on the display by a "begin" marker which is inserted by TED into the text. The marker is displayed as a graphic left bracket (character value B7H). The ending position will be indicated on the display by an "end" marker which is also inserted by TED into the text. The marker is displayed as a graphic right bracket (character value BBH). These markers occupy ordinary text positions; thus they may be deleted or overstriked. Any remaining in the text buffer at the time a FILE command is performed will be written to the disk file just as if they were ordinary text characters [internally, TED uses the value FEH for a "begin" marker and the value FFH for an "end" marker].

Although you can mark as many blocks as your heart desires, TED provides no way to differentiate between marked blocks in other than the BLOCK-DELETE

function. For copying and moving blocks, the first block marked in the text is the one chosen for copying or moving. On the other hand, a BLOCK-DELETE request requires that the cursor be positioned within the interior of the marked block which is to be deleted.

To COPY the first marked block in the text to some other position, simply mark the beginning and end of the block as discussed above, move the cursor to the position in the text where you want the marked block copied into, then invoke the block copy command via the sequence, <^B> followed by <C>. Note that the block which will be copied is the first marked block found in the text buffer. A few things could go wrong with your request. If TED can find no properly marked block, it will display the error message,

Marker!

and terminate the block mode. Another error which could occur is when the position you wish the block copied into happens to be in the interior of the block itself! Such a block copy can not occur. You will be informed of this by a display of the error message,

Cursor!

The successful block copy operation only copies the marked text; the markers are not copied as well. In fact, the marked text remains in its original position relative to the text which surrounds it. The cursor position relative to the text will be unchanged after the block is copied; however, the screen may be refreshed and the physical location of the cursor on the screen may be different.

A block of text may be MOVED from one position to another by a command sequence similar to the block copy. In this case, simply mark the beginning and end of the block as discussed above, move the cursor to the position in the text where you want the marked block moved to, then invoke the block MOVE command via the sequence, <^B> followed by <M>. Again note that the block which will be moved is the first marked block found in the text buffer. This operation is essentially one of copying and automatic deleting without the double check prompt. As in the case of the block copy, the same errors are possible with similar diagnostic messages when things are not as they should be. With the block move command, the new cursor position will be the new position of the moved block. The screen may be refreshed and the physical cursor position altered to accomodate this request.

The last block operation is deletion. Similar to the above functions, you first must mark the block's beginning and ending positions. You must then position the cursor to the interior of the marked block and invoke the command with the sequence, <^B> followed by <D>. If TED is confident that the cursor position is interior to a marked block, it will double check your request by issuing the prompt,

Sure?

It is necessary to depress <ENTER> to affirm your intentions. Any other character entry (including a "Y") will cause TED to ignore the block delete request.

The same errors as for copy and move can occur; however, the messages may not be for the same reasons. When a block delete is requested, TED will first look for an ending block marker starting from the cursor position. If none is found, the error displayed will be "Marker!". This doesn't mean necessarily that a properly marked block is missing. On the other hand, if an ending marker is found past the cursor position, TED next scans forward for a beginning block marker. A "Marker!" error will also be posted if none is found. If a marker is found but is also past the cursor position, a "Cursor!" error will be posted. The remaining situation is the correct one; the cursor is positioned interior to the marked block and that block will be deleted. The screen will be refreshed and the cursor will be moved to the relative position of the block which was deleted.

Filing away your text to a disk file

The <^F> command is used to FILE the contents of the text buffer area into a disk file. When you depress <^F>, you will be prompted for the name of the file via the query message,

Filespec?

If the file specification you wish to use has an extension of "/TXT", you do not have to enter the extension. If the extension is omited from your entry, "/TXT" will be automatically provided.

The FILE command will save the entire text buffer, excluding the terminating NULL but including any block markers, into the disk file identified by your input. If any disk write error is encountered while saving the text buffer into the disk file, the message,

I/O error!

will be displayed. In any case, the text buffer is left undisturbed.

Text search

TED provides the SEARCH command to scan the text buffer for a specified string of characters. You specify the search by invoking the command with <^S>. TED then prompts you for the search string with the query message,

String?

You can enter up to 23 characters to be used for the search string. Terminate your search string with an <ENTER> (the <ENTER> character code is not included as one of the 23 characters). TED will then look for the string

starting with the first character immediately following the cursor. The matching is case sensitive which means that characters entered in upper case must be found in upper case and characters entered in lower case must be found in lower case. If the search string cannot be found, the message,

Can't!

will be displayed. At this point, the cursor location remains unchanged. If, on the other hand, a matching string of text is found in the text buffer, it will be displayed. The display window will be redrawn starting with the line which contains that string. The cursor will be repositioned to the first character of the matching string.

If you enter a null string in response to the "String?" query, then the search will proceed with the last entered non-null search string, providing one was available. A null string is, of course, entered by responding to the "String?" query with an immediate <ENTER> keystroke. Using this procedure, you can advance the cursor to each occurrence of the search string in question.

Another way to find each occurrence of a search string is with the GO command, <^G>. Each depression of <^G> is identical to the sequence, <^S> followed by <ENTER>.

Text search and replace

TED also provides the capability of replacing a text string matching up with the search string with a different string - the replacement string. When the REPLACE command is invoked via <^R>, the query message,

String?

will be displayed. Although the message is the same as for SEARCH, this query is asking you for the replacement character string. You can enter up to 23 characters to be used for the replacement string. Terminate your string with an <ENTER> (the <ENTER> character code is not included as one of the 23 characters). TED will then look for the currently pending SEARCH string starting with the first character IMMEDIATELY under the cursor. If the SEARCH string cannot be found, the message,

Can't!

will be displayed. At this point, the cursor location remains unchanged. If, on the other hand, a matching string of text is found in the text buffer, it will be replaced with the REPLACE string. The display window will be redrawn starting with the line which contained the string which was replaced. The cursor will be repositioned to the first character immediately following the replacement string.

If you wish to replace the next occurrence of text which matches up with the SEARCH string with that same REPLACEMENT string, all you need do is invoke the REPLACE command, <^R> followed by a NULL replacement string. A null string is, of course, entered by responding to the "String?" query with an immediate <ENTER> keystroke. Since the cursor is repositioned after a replacement to the first character immediately following the replacement string, your subsequent replacements will not get you into trouble if the SEARCH string happened to be a substring of the REPLACEMENT string!

The GO command, <^G>, still functions to find the next occurrence of the SEARCH string. Knowing this, if the next occurrence of the search string is beyond the text currently displayed on the screen and you wish to confirm its replacement, simply GO to the next occurrence then REPLACE, as necessary.

Printing text

TED provides no facility for printing your text file. The most reasonable way of accomodating that function is with the LIST command provided as part of your DOS. Remember, the LIST command can be invoked via the PRO-NTO LIBEXEC facility.

Text recovery with OOPS

There may be times when you exit the TED application inadvertantly without filing the edited text to a disk file. Some full-featured word processors permit you to re-enter their editor with an asterisk parameter which generally denotes "reclaim the text buffer". Since TED is invoked via a PRO-NTO function key, there is no way to specify such a parameter. However, to provide this feature, another application is included called, OOPS. The OOPS application is identical to TED with one important difference. Instead of automatically clearing the text buffer as TED does, OOPS will display whatever is in the external memory bank which it obtains. Thus, if you have not altered any of the information in that memory bank, you can always go back and recapture it with OOPS.

One cautionary note. Since all of the text pointers normally established by TED will not be initialized when invoking OOPS, it will be necessary to scroll through the text until reaching its last character prior to doing any other operation. This may also be performed using NEXT PAGE.

Alterations to TED

You can alter certain characteristics of TED via direct modification using the File EDitor, FED. FED is one of the editors included with this package of applications. All of the following "Byte" references denote a position within the sector numbered "1" of the TED/APP program file. Codes in the tables below are the hexadecimal values provided by TED.

The cursor characters may be changed in the following positions:

Byte	Code	Cursor
02	<5F>	overstrike
03	<BF>	insert

The editing command keys may also be changed to suit your preference. Please note that when making any changes, avoid conflicts in keystrokes. For example, a <CTRL-X> and a <SHIFT LEFT ARROW> both generate the same code, <18>. Thus, if you choose to utilize <CTRL-X>, then you can't use <SHIFT LEFT ARROW>. You also cannot use any of the FUNCTION keys (F1, F2, F3) for TED commands as PRO-NTO is in an active mode during TED's operation and therefore PRO-NTO has control over the function keys.

Byte	Code	Command
04	<08>	LEFT
07	<09>	RIGHT
0A	<0A>	DOWN
0D	<0B>	UP
10	<0D>	ENTER
13	<18>	BOL
16	<19>	EOL
19	<1A>	END
1C	<1B>	TOP
1F	<01>	INSERT
22	<02>	BLOCK
25	<04>	DELETE
28	<06>	FILE
2B	<07>	GO
2E	<0C>	LOAD
31	<0E>	NEXT page
34	<12>	REPLACE
37	<13>	SEARCH
3A	<15>	UP page
3D	<BD>	EXIT

Finally, it was noted that TED reverts to overstrike mode when it is first invoked. This can be changed to insert mode by the following change:

Byte: B8 Overstrike Code: <81> Insert Code: <80>

This page intentionally left blank

VED Application

The VED application will allow you to edit the 80 column by 24 row video screen display. You can edit the screen image existing at the time VED is invoked; save the current image to a screen image disk file; or load a screen image disk file into the video screen. The altered video image or a portion of that image may then be exported back into the program which you were running when VED was invoked.

When you invoke the VED application, it will request PRO-NTO to open a 24 row by 80 column window. If the window cannot be opened, a short beep will sound from your computer's internal speaker and VED will terminate. When the window is opened, VED will restore the previous screen image to the window and place you into the EDIT mode. You will see a momentary flash of the screen.

The EDIT mode enables you to alter the text which appears on the screen. When you are finished editing, there are two ways of terminating the edit mode. One way is to enter <CONTROL-F> to "finish" the edits and go into the EXPORT mode. The other way is the depress the <BREAK> key which will abort the editing and return you to the previously running program.

Summary of VED's editing commands

Keystroke	Editing operation
<LEFT ARROW>	Move the cursor left one column.
<RIGHT ARROW>	Move the cursor one column to the right.
<DOWN ARROW>	Move the cursor down one line.
<UP ARROW>	Move the cursor up one line.
<SHIFT><LEFT ARROW>	Move to the start of the line.
<SHIFT><RIGHT ARROW>	Move the cursor to the last column.
<SHIFT><DOWN ARROW>	Move the cursor to the last line.
<SHIFT><UP ARROW>	Move the cursor to the top line.
<CONTROL-A>	Add a space and push rest of line right.
<CONTROL-B>	Mark the beginning of a cut & paste block.
<CONTROL-C>	Concatenate the next line to overstrike the position of the cursor.
<CONTROL-D>	Delete the character under the cursor and move the trailing portion of the line left.
<CONTROL-E>	Mark the end of a cut & paste block.
<CONTROL-F>	Write the screen contents to a disk file.
<CONTROL-L>	Load the screen from a disk file.
<CONTROL-S>	Split this line at the cursor and move the trailing text to a NEW next line.
<CONTROL-T>	Designate where to paste the marked block.
<BREAK>	Cancel the changes and exit.
<ENTER>	Move the cursor to the beginning of the next line.
<CLEAR><RIGHT ARROW>	Terminate edits and enter EXPORT mode.

Text entry

A blinking cursor which appears as a plus-minus sign (a character value of 7FH - see CHARSET) will appear in the first character position of the text area. All text is entered in what is termed overstrike mode. Any ASCII non-control character will overtype the character beneath the cursor and cause the cursor to advance by one position.

You can insert a character by first moving the cursor to the insertion position and then depressing <CONTROL-A> (for "add" of a character). The characters extending from that position through to the end of the line will be shifted right and a space will be inserted. You can then overstrike the space with the desired character.

Text deletion

To delete a character, depress <CONTROL-D> (for "delete" a character). The character at the cursor position will be deleted and all trailing characters in the line will be shifted left to "take up the slack".

Combining two text lines

The line concatenate operation invoked by <CONTROL-C> will overstrike the characters from the cursor to the end of the line with the text which appears on the next line. The entire next line will be deleted and all lines to the bottom of the text will be bumped up by one line. If the next line is longer than the character positions remaining, the "overflowed" characters will be dropped. If you are aware that a concatenation of the next line will overflow and you do not want this to happen, you may want to split the next line into two pieces. See the split command, <CONTROL-S>.

Splitting a line into two lines

The line split operation invoked by <CONTROL-S> will divide up the current line into two lines. This is done by first moving all lines following the current line down by one line which opens up a blank line. Next, all of the characters from the cursor through to the end of the current line are shifted to the new blank line. The old last line of the text will be saved in an overflow line buffer so that the next "concatenate" operation will pull it back into the visible text area.

Cutting and pasting a block of text

The cut and paste facility allows you to move a designated portion of text to another position on the screen. VED operates by cutting out the marked block and pasting it to the screen where you designate. The actual placement of the pasted text and the text impacted by the placement may be

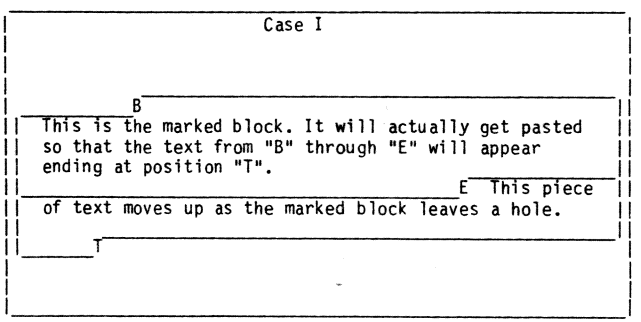
adjusted so as to not overwrite any text on the screen. In any case, the relative positioning of the pasted text will be according to your specification.

You mark the beginning character of the text block by first positioning the cursor to appear over the first character of the block and then depressing <CONTROL-B>. You will note the BEGIN marker by the appearance of a left bracket "[" character blinking over the character which was at the cursor. If a BEGIN marker was already on the screen at some other position, it will be relocated to the current cursor position. If the current cursor position already contained a BEGIN marker, it will be removed. The cursor will be advanced by one position.

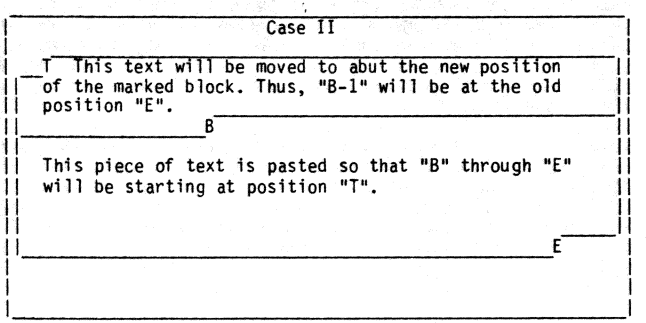
You mark the ending position of the text block by first positioning the cursor to appear over the last character of the block and then depressing <CONTROL-E>. You will note the END marker by the appearance of a right bracket "]" character blinking over the character which was at the cursor. If an END marker was already on the screen at some other position, it will be relocated to the current cursor position. If the current cursor position already contained an END marker, it will be removed. The cursor will be advanced by one position.

Both the BEGIN and END markers may be repositioned at will. The marked text constitutes all characters between and including the marked endpoints as if the end of one row was connected to the beginning of the next row. The cut and paste operation is invoked by depressing <CONTROL-T> after you have positioned the cursor where the marked text is to be pasted. If the END marker precedes the BEGIN marker or the PASTE position is interior to the marked block, the pasting will not be performed.

Since the result of this cut and paste facility may be hard to visualize, the following two diagrams may be studied to see this behavior. Also, it will be beneficial to try out the cut and paste operation on sample text. Remember, if you exit VED via the <BREAK> key, nothing is disturbed.



The "Case I" example illustrates the paste position after the marked block. The positions "B" and "T" designate the endpoints of the text which is disturbed. The text actually is rotated counter clockwise so that the marked text immediately follows the text between the end of the marked position and the paste position. Thus, the piece of text between "E+1" and "T" can be conceptualized as the marked text to be pasted into position "B".



In "Case II", the marked text "B" through "E" will be pasted into the position designated as "T"; however, the text from "T" through "B-1" will be rotated to follow the new position of "E".

Saving the screen to a disk file

If you wish to save the current contents of the video screen to a disk file, you may do so by depressing <CONTROL-F>. A small window will open up to prompt you for the file specification. VED will save the contents of the video screen as 24 lines of 80 characters; each line will be terminated by a carriage return (the <ENTER> key code). This is the identical format used by the DOSAVE and DOLOAD applications. After the screen is saved to disk, the blinking cursor will reappear and you may continue your edits.

Loading the screen from a disk file

If you wish to load some other previously saved video screen from a disk file, you may do so by depressing <CONTROL-L>. A small window will open up to prompt you for the file specification. VED will load the contents of the file into the video screen overwriting the previous image. The file must be a video screen format file as generated by DOSAVE, VED, or other application. After the screen is loaded from disk, the blinking cursor will reappear and you may continue your edits on this new image.

Exiting from VED

When you have completed your edits and wish to export the screen (or a portion of it) back to the interrupted program, depress <CLEAR RIGHT ARROW>. VED will then enter the EXPORT mode. Don't forget that if you abort the editing by depressing the <BREAK> key, VED will return to the previously interrupted program as if VED had not been invoked.

There are two ways of controlling what PRO-NTO does at the end of each exported line depending on how you mark the closure of the rectangle. If you close the rectangle via the <ENTER> key, a carriage return will be added to the "input" at the end of each marked line. This carriage return will be appended regardless of whether the marked rectangle is one or more lines. If the rectangle is closed by the depression of <SHIFT ENTER>, then the line is input from the beginning mark to the ending mark in a continuous stream; no carriage returns are added by PRO-NTO.

The rectangle is defined by the two points making up its northwest to southeast diagonal. These two points may be marked in the following manner:

1. Position the cursor to the upper left corner of the rectangle which will contain the information. The four arrow keys, <LEFT>, <RIGHT>, <UP>, and <DOWN>, will move the cursor around the screen.

2. Depress <CONTROL><B> to mark the beginning of the rectangle block. The character under the cursor will be replaced on the screen with a left square bracket which indicates the marked position. Don't worry about the bracket displayed; the correct character will be provided as input.

3. Position the cursor to the lower right corner of this rectangle again using the four arrow keys. This position may be on the same display row as the beginning mark.

4. Depress the <ENTER> or <SHIFT ENTER> key to mark the end of the marked block. This now defines the rectangle. The export will commence. The export function may be aborted anytime prior to marking the end of the rectangular block simply by depressing the <BREAK> key.