Please do not upload this copyright pdf document to any other website. Breach of copyright may result in a criminal conviction.

This Acrobat document was generated by me, Colin Hinson, from a document held by me. I requested permission to publish this from Texas Instruments (twice) but received no reply. It is presented here (for free) and this pdf version of the document is my copyright in much the same way as a photograph would be. If you believe the document to be under other copyright, please contact me.

The document should have been downloaded from my website https://blunham.com/Radar, or any mirror site named on that site. If you downloaded it from elsewhere, please let me know (particularly if you were charged for it). You can contact me via my Genuki email page: https://www.genuki.org.uk/big/eng/YKS/various?recipient=colin

You may not copy the file for onward transmission of the data nor attempt to make monetary gain by the use of these files. If you want someone else to have a copy of the file, point them at the website. (https://blunham.com/Radar). **Please do not point them at the file itself as it may move or the site may be updated.**

It should be noted that most of the pages are identifiable as having been processed by me.

I put a lot of time into producing these files which is why you are met with this page when you open the file.

In order to generate this file, I need to scan the pages, split the double pages and remove any edge marks such as punch holes, clean up the pages, set the relevant pages to be all the same size and alignment. I then run Omnipage (OCR) to generate the searchable text and then generate the pdf file.

Hopefully after all that, I end up with a presentable file. If you find missing pages, pages in the wrong order, anything else wrong with the file or simply want to make a comment, please drop me a line (see above).

It is my hope that you find the file of use to you personally – I know that I would have liked to have found some of these files years ago – they would have saved me a lot of time !

Colin Hinson
In the village of Blunham, Bedfordshire.

TEXAS INSTRUMENTS

TERMINAL EMULATOR PROTOCOL MANUAL

May 18, 1981

IMPORTANT NOTICE REGARDING TECHNICAL DATA

TEXAS INSTRUMENTS MAKES NO WARRANTY, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THIS LITERATURE OR ANY INFORMATION DERIVED THEREFROM AND MAKES ALL MATERIAL AVAILABLE SOLELY ON AN "AS IS" BASIS.

IN NO EVENT SHALL TEXAS INSTRUMENTS BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING OUT OF THE PURCHASE OR USE OF THIS LITERATURE AND THE SOLE AND EXCLUSIVE LIABILITY OF TEXAS INSTRUMENTS, REGARDLESS OF THE FORM OF ACTION, SHALL NOT EXCEED THE PURCHASE PRICE OF THIS BOOK. MOREOVER, TEXAS INSTRUMENTS SHALL NOT BE LIABLE FOR ANY CLAIM OF ANY KIND WHATSOEVER BY ANY OTHER PARTY AGAINST THE USER OF THIS LITERATURE.

Table Of Contents

SECTION 1

INTRODUCTION

SECTION 2

THE TI-99/4 ENVIRONMENT

SECTION 3

CODED DATA

SECTION 4

EMULATOR CONTROL FORMATS

SECTION 5

FILE TRANSFER

APPENDIX

SECTION 1

INTRODUCTION

1.1 PURPOSE

This manual provides a complete description of the communication protocol used by the terminal emulator packages (starting with Terminal Emulator II). It describes the steps needed to display text, create and display graphics and create and execute sound and speech on the TI-99/4 using the terminal emulator protocols.

1.2 AUDIENCE

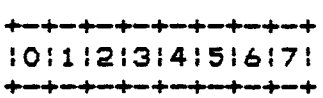
This manual is designed for programmers of host systems charged with writing software to interface with the terminal emulator packages.

1.3 CONVENTIONS

The following conventions will be observed:

1. Hex numbers will be denoted with a greater than symbol (>). For example hex 20 will be written as >20.

2. Bits will be numbered from left to right. For example the bit numbering for a byte would be:



SECTION 2

THE TI-99/4 ENVIRONMENT

2.1 INTRODUCTION

Effective use of the terminal emulator protocols requires a working knowledge of the TI-99/4. This section provides information about video control, sound, speech and file access needed to implement the protocols on a host system. Later sections describe the actual format of the protocols.

2.2 VIDEO CONTROL

The terminal emulator supports the graphics and text modes of video operations. The graphics mode will be explained followed by the differences between graphics and text modes.

2.2.1 GRAPHICS MODE. The TI-99/4 supports 256 different characters for screen display. The characters are identified by numeric codes >0 to >FF. Codes in the range >20 to >7E are supplied a default definition by the terminal emulator. For example, code >41 is defaulted to the letter A. (See appendix A for a complete list of defaults.) The user can redefine any of the 256 characters by using the protocols (see section 4.3.1).

NOTE

Any of the 256 characters can be given a definition by the user. However, not all the characters can be displayed by the emulator package. Characters >00, >0A, >0C, >1B, >80, >8A, >8C and >9B cannot be displayed by the emulator in graphics mode.

2.2.1.1 The Screen Display. In graphics mode the screen is divided into 24 lines with 32 characters on each line. Thus, there are a total of 768 character positiions on the screen. The

lines are numbered from 0 to 23 and the columns are numbered from 0 to 31.

The terminal emulator maintains a logical cursor position on the screen. The cursor symbol is not displayed while in graphics mode. The next character received will be placed on the screen at the position identified by the cursor. The cursor is then advanced to the next screen postion. To display a character at the given cursor position the character code is transmitted to the TI-99/4. For example, to display the default character specified by code >41 (the letter A) on the screen, the host system would transmit the >41 to the emulator.

The protocol lets the user position the cursor on the screen. Thus the host could display one character at line 10 column 6 and the next character at line 0 column 0. (See section 4.2).

2.2.1.2 Character Banks. Standard data transmission allows for 7 bits of data and 1 bit to check for parity. Because of the parity bit the maximum number that can be transmitted to the TI-99/4 is >7F. Using standard transmissions the host cannot display characters in the range >80 to >FF.

To overcome this limitation the terminal emulator has divided the characters into two banks for display purposes. Bank 1 (or the lower bank) contains the characters >0 to >7F and bank 2 (or the upper bank) contains the characters >80 to >FF. By default the terminal emulator will use bank 1 to display characters. The protocols, however, contain a command sequence to let the host switch between banks.

When the emulator is using the upper bank, >80 is added to the received character code before the code is placed on the screen. For example, suppose the user wants to display >41, >80 and >41. Since the emulator defaults to the lower bank the user transmits >41 for the first character. The character >80 is in the upper bank (>80 to >FF) so the host must send the command sequence to switch banks. The host can send either >80 or >30 (>80 - >80) for the second character. Since the parity bit is always removed by the emulator, the byte will be >30 when the emulator is ready to display it.

NOTE

Some host systems may put limitations on the transmission of characters in the range >80 to >FF. Be certain the constraints of the host system are understood when designing

programs. Regardless of what transmissions the host allows, all bytes in the range >80 to >FF will be converted to the range >0 to >7F when the parity bit is removed.

Since the emulator is now operating on the upper bank, >80 will be added to the received byte and >B0 (>30 + >80) will be displayed.

The final character to send is >41 which is in the lower bank (>0 to >7F). The user must send the control sequence to switch banks so the emulator will once again be operating on the lower bank. The character can then be transmitted. (See section 4.3.5).

2.2.1.3 Character Definition. There are 768 character positions on a standard graphics screen. Each pattern position is composed of an 8-by-8 grid of dots called pixels. Each pixel in the character grid can be either "on" or "off". A character definition simply defines the status (on or off) of each of the 64 pixels in a character grid.

To define a character using the terminal emulator protocols the user specifies the code of the character to be defined (>0 to >FF) and eight bytes of hex data. There are a total of 64 bits in the eight bytes of data. Each of the bits corresponds to one of the pixels in the character grid. The eight bits in byte one refer to the first row of eight pixels, the eight bits in byte two refer to the second row of 8 pixels and so on.

If a bit in the character definition contains a 1 the corresponding pixel is turned on. If a bit contains a 0 the corresponding pixel is turned off. Figure 2-1 is an example of the hex data required to define the letter T. (See section 4.3.1).

	Character Grid									Hex Data	
Row 1	1	1	1	1	1	1	1	1		Byte 1	FF
Row 2	1	1	1	1	1	1	1	1		Byte 2	FF
Row 3	0	0	0	1	1	0	0	0		Byte 3	18
Row 4	0	0	0	1	1	0	0	0		Byte 4	18
Row 5	0	0	0	1	1	0	0	0		Byte 5	18
Row 6	0	0	0	1	1	0	0	0		Byte 6	18
Row 7	0	0	0	1	1	0	0	0		Byte 7	18
Row 8	0	0	0	1	1	0	0	0		Byte 8	18

Figure 2-1 LETTER T DEFINITION

2.2.1.4 Defining Character Color. The TI-99/4 supports 16 different colors on the video display. However any given character can only contain two colors called the foreground color and the background color. The colors are denoted by numeric codes ranging from >0 to >F. Table 2-1 lists the colors and hex codes supported by the emulator.

Table 2-1 COLOR CODES

Hex	Color
>0	Transparent
>1	Black
>2	Medium Green
>3	Light Green
>4	Dark Blue
>5	Light Blue
>6	Dark Red
>7	Cyan
>8	Medium Red
>9	Light Red
>A	Dark Yellow
>B	Light Yellow
>C	Dark Green
>D	Magenta
>E	Gray
>F	White

Each pixel in the character grid on the video can be either the foreground or the background color. The bits in the character definition specify the color. If the bit contains a 1 the corresponding pixel uses the foreground color (the pixel is "on"). If the bit contains a 0 the corresponding pixel uses the background color (the pixel is "off").

For color control the 256 characters are divided into 32 groups called color sets. Each color set contains eight characters. Table 2-2 gives the color sets.

Table 2-2 COLOR SETS

Color Set	Characters
> 0	>00 to >07
> 1	>08 to >0F
> 2	>10 to >17
> 3	>18 to >1F
> 4	>20 to >27
> 5	>28 to >2F

> 6	>30 to >37
> 7	>38 to >3F
> 8	>40 to >47
> 9	>48 to >4F
> A	>50 to >57
> B	>58 to >5F
> C	>60 to >67
> D	>68 to >6F
> E	>70 to >77
> F	>78 to >7F
>10	>80 to >87
>11	>88 to >8F
>12	>90 to >97
>13	>98 to >9F
>14	>A0 to >A7
>15	>A8 to >AF
>16	>B0 to >B7
>17	>B8 to >BF
>18	>C0 to >C7
>19	>C8 to >CF
>1A	>D0 to >D7
>1B	>D8 to >DF
>1C	>E0 to >E7
>1D	>E8 to >EF
>1E	>F0 to >F7
>1F	>F8 to >FF

All characters in a given color set are assigned the same foreground and background colors. For example, assume character >23 has a foreground color of black and a background color of cyan. Then all characters in color set >4 (characters >20 to >27) will have the same foreground and background colors. The only way characters can have different colors is for them to be from different color sets.

One byte is required to define the color for a color set. Numbering bits 0 to 7 from left to right, bits 0-3 contain the foreground color and bits 4-7 contain the background color. For example, assume color set >B is to have a foreground color of black and a background color of cyan. From Table 2-1 we know black has a color code of >1 and cyan has a color code of >7. The byte required to define the color would be >17.

The host system can define the foreground and background colors for the characters using the protocols. The host must supply the color set identifer (>0 to >1F) followed by the desired foreground and background colors. (See section 4.3.6).

2.2.1.5 Defining Screen Color. The TI-99/4 also lets a user assign a color to the entire screen. This includes the area at the top and bottom of the screen, called the border, where characters cannot be placed. The screen color is the color displayed when transparent (color code >0) is used as a character color. The protocols supports a command sequence that lets the host define the screen color. (See section 4.3.12).

2.2.2 TEXT MODE. Text mode differs from graphics mode in the following ways:

1. Each line on the screen is 40 characters long. Thus there are 960 character positions on the display.

2. Each character grid on the screen consists of six columns and eight rows of pixels for a total of 48 pixels per character. Eight bytes of data are still required to define the grid. However, the last two bits in each byte are ignored by the system.

3. All characters are limited to the same foreground and background colors. A special command sequence is used to change the colors. The same command sequence will change the screen color (refer to 2.2.1.5) when the emulator is in graphics mode.

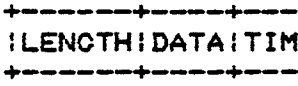
4. Characters >00 to >1F, >7F, >80 to >9F and >FF will not be displayed while in text mode.

The protocol has a special command sequence used to switch between text and graphics modes. (See section 4.2).

2.3 SOUND

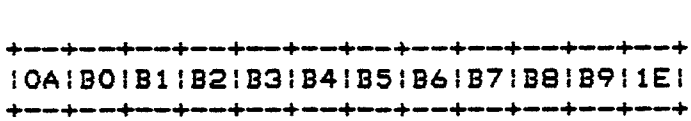
The TI-99/4 supports three sound generators and one noise generator. The terminal emulator protocols let the host system control the generators. The generators can be used to create music and sound effects.

2.3.1 SOUND LIST. The volume and frequency of the generators are controlled by a memory resident table called a sound list. The sound list is composed of variable length control blocks. The general format of a control block is:



LENGTH is a one byte field that specifies the number of bytes contained in DATA. DATA is a variable length area that controls the volume and frequency output of the generators. TIME is a one byte field that specifies the time interval the TI-99/4 is to wait before it starts processing the next control block. The timer is in 60ths of a second.

For example, assume that the user has defined ten bytes of generator control information (a complete explanation of this area is below). The TI-99/4 is to process a block with the ten bytes of data and pause for 1/2 second. The length byte (LTH) would contain >A to specify that ten bytes of control information follows (DATA). Next would come the actual control information. And finally the timer byte (TIME) would contain a >1E to specify that a pause of 1/2 second (30/60ths of a second) is to occur before processing the next control block. The sound will remain constant until the next control block is processed. The actual sound list would be:



A sound list on the TI-99/4 can be terminated in two ways. One option is to stop processing the sound lists completely. To do this the timer byte of the last control block within the list must contain a zero. The control block associated with the zero timer byte will be processed.

NOTE

Once a generator has been started it will continue to output sound at a constant volume and frequency until a control block in a sound list modifies the settings. Unless the programmer wants a sound to continue at a constant volume and frequency, the control block that terminates the sound list (control block with a timer byte of zero) should turn off all generators by setting all volumes to off.

The other termination option is to use the last entry in a sound list to branch to the start of the same sound list or to the start of another sound list. Using this method the host can produce continuous sound with a limited amount of control data. To link to another sound list (or to the start of the same list) specify a length byte of zero followed by the two byte address of the next sound list. The protocols have a command sequence to turn off all sound generators. This provides a means of stopping continuous sound. (See section 4.3.4).

The emulator package has set aside 16 table areas in memory to hold sound lists. The tables are numbered >0 to >F. Each table is 128 bytes in length and the areas are contiguous in memory. The first table (number 0) is at address >1100, the second table (number 1) is at >1180 and so on. The host can use the protocols to store data in the tables. (See section 4.3.2).

A sound list may be larger than 128 bytes. If it is larger than 128 bytes it will use part of the next table in sequence to hold the excess. For example, assume table >3 is loaded with a 224 byte sound list. Then all of table >3 is used and 96 (224 - 128) bytes of table >4 are also used. If the user tried to load another sound list into table >4 he would destroy the last 96 bytes of the sound list loaded in table >3.

Assume the host wants to place a sound list in table >2 and link it to play music continuously. The host constructs the sound list and loads it into table >2. The last control block in the list would contain a length byte of zero followed by address >1200 (memory address of sound table >2).

2.3.2 SOUND LIST CONTROL BLOCK. The control block specifies three items of information: volume level for sound and noise generators; frequency for sound generators and frequency for the noise generator. The information can be specified in any sequence and for any number of generators.

2.3.2.1 Volume Level. Eight bits of information are required to control volume levels. The bits are numbered 0 to 7 from left to right. The format of the bits is:

1. bit 0 - Must be set to 1.

2. bits 1-2 - Denotes the generator to control. Valid entries are:

   a. 00 - Sound generator 0.

   b. 01 - Sound generator 1.

c. 10 - Sound generator 2.

d. 11 - Noise generator.

3. bit 3 - Set to 1 to indicate volume control.

4. bits 4-7 - Volume setting to use. Possible values (in binary) range from 0000 to 1111. 0000 is the highest volume, 1000 is a medium volume and 1111 turns the generator off.

For example to turn generator 1 to the highest volume the volume level byte would be: 1 01 1 0000. To turn the generator 0 to a medium volume the volume level byte would be: 1 00 1 1000. To turn generator 3 (noise generator) off the volume level byte would be: 1 11 1 1111.

2.3.2.2 Sound Generator Frequency Control. Two bytes are required to control the fequency of the sound generators. The bits are numbered 0 to 15 from left to right.

The frequency is converted to a value called the frequency count. The count is then specified in the control byte instead of the original frequency. The count is always specified as ten bits of data. The formula used for the conversion is COUNT = 111860 / FREQUENCY. For example, the count for a frequency of 110 Hz would be 111860 / 110 which is 1016.9. Since the count must be an integer this rounds to 1017.

NOTE

The frequency range allowed is 110 Hz to 55,938 Hz, or a count of 1017 to 2.

The format of the frequency bytes is:

1. bit 0 - must be set to 1.

2. bits 1-2 - Generator to work with. Valid entries are:

a. 00 - Sound generator 0.

b. 01 - Sound generator 1.

c. 10 - Sound generator 2.

NOTE

The noise generator (bit setting 11) is not a valid entry.

3. bit 3 - Set to 0 to indicate frequency control.

4. bits 4-7 - The LAST four bits of the count.

5. bits 8-9 - Set to 00.

6. bits 10-15 - The FIRST six bits of the count.

As an example assume that generator 1 is to be set to a frequency of 5,676 Hz. First the frequency must be converted to the count. Using the formula specified in 2.3.2.2 gives: COUNT = 111860 / 5676 = 19.7. The count is then rounded to 20. Next the count must be converted to 10 bits. This gives: 000001 0100. Remember, the last 4 bits (0100) go into bits 4-7 and the first 6 bits (000001) go into bits 10-15.

The full 16 bits required to set the frequency for the above example is:

1 10 0 0100 00 000001

2.3.2.3 Noise Frequency Control. Eight bits are required to control the frequency of the noise generator. The bits are numbered 0 to 7 from left to right. The format of the byte is:

1. bit 0 - Must be set to 1.

2. bits 1-2 - Generator to work with. MUST be set to 11 for the noise generator.

3. bit 3 - Set to 0 to indicate frequency control.

4. bit 4 - Must be set to 0.

5. bit 5 - Type of noise indicator. 0 = White noise, 1 = Periodic noise.

6. bits 6-7 - Quality of noise. Valid entries are 00, 01, 10 and 11. If 11 is used the noise is dependent on the frequency specified for generator 3.

Different "qualities" of noise can be produced by varying bits 6-7. The best approach is to try the different values allowed and observe the results. Experiment with bits 6-7 as 00, 01 and 10. Also set bits 6-7 to 11, try different counts for generator 3 and observe the different "qualities" of noise.

2.4 SPEECH

Dictionary speech and text-to-speech are two forms of speech supported by the emulator protocols. A speech synthesizer must be connected to the TI-99/4 before either form of speech can be used.

2.4.1 SPEECH DICTIONARY. The speech peripheral that attaches to the TI-99/4 has a list of words it can speak called the speech dictionary. These words are created by recording the words and converting the recording to linear predictive code. The data is then stored in the peripheral. The speech peripheral manual contains a complete list of the words in the dictionary. The protocols let the host speak the words from the dictionary.

2.4.2 TEXT-TO-SPEECH. The emulator also supports the text-to-speech capability which can be used in two ways. One method is to use protocol commands and transmit text strings to the emulator. The emulator will use rules of grammar to convert the text string to allophones. The allophones will then be spoken. Any text string can be spoken using text-to-speech.

The other method is to use protocol commands to transmit the allophones themselves to the emulator. The terminal emulator command module manual explains allophones. (See sections 4.3.7 to 4.3.11).

2.5 DISK FILES

The diskettes used by the TI-99/4 are divided into sectors. Each sector is 256 bytes in length. Disk space is allocated to files by whole sectors. A file that is 257 bytes long is allocated two disk sectors. Only one byte out of the second sector will be used.

The TI-99/4 supports three types of disk files. They are:

1. Fixed Length Record.

2. Variable Length Record.

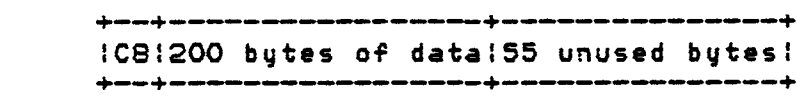
3. Basic Program Image.

2.5.1 FIXED LENGTH RECORD FILES. At file creation time the user specifies the length of the data records. All records in the file have the same length. File management will group the records into disk sectors.

For example, assume a file has been created with 80 byte records. Each disk sector will contain three records. The three records use 240 bytes (3 times 80) of the 256 bytes in the sector. The remaining 16 bytes of the sector are not used.

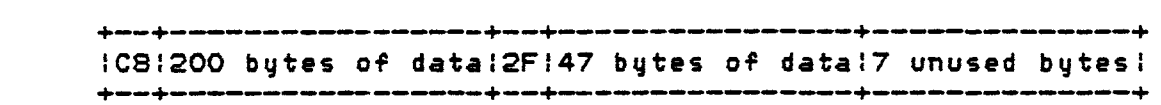
2.5.2 VARIABLE LENGTH RECORD FILES. At file creation time the user specifies a maximum length for the data records. The format of the data on disk is one byte containing the actual length of the data followed by the data. The maximum length allowed is >FE (254 decimal). File management will put as many records as possible in a disk sector. A length of >FF denotes the end of data in a sector.

For example, a user has created a variable length file with a maximum record size of 250 bytes. The user has already written one record to the file containing 200 bytes of data. The format of the disk sector is:



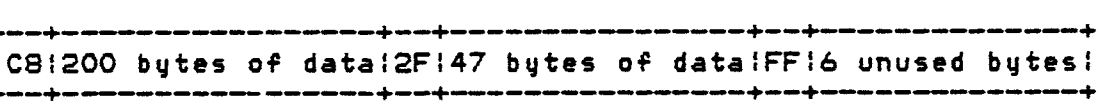
Thus 201 bytes of the sector (one byte for >C8 (decimal 200) denoting the record length and 200 bytes for the data) have been used.

Next, the user writes a 47 byte record to the file. The format of the sector is now.



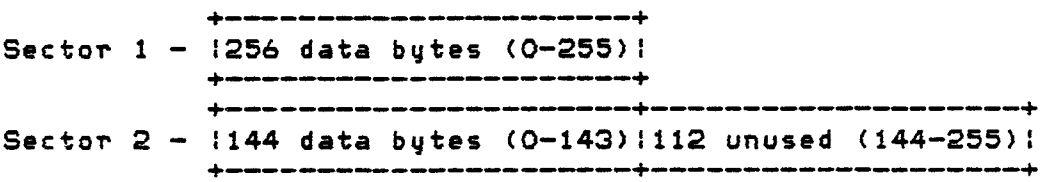
Now 249 bytes of the sector (201 bytes for the first record, one byte for >2F (47 decimal) denoting the length of the second record and 47 bytes for the second record's data) have been used.

Finally, the user writes a 50 byte record to the file. File management requires 51 bytes to write the record to disk (one byte for the length and 50 bytes for the data). The current sector, however, only has 7 unused bytes remaining. The end of sector marker (>FF) is placed in the current sector and the third record is written to the next sector. The disk sector after the write is:



2.5.3 BASIC PROGRAM IMAGE FILES. File management starts with the first sector in a program file and completely fills each sector with data until all program data is saved. The last sector in the file may not be completely full. File management maintains a pointer to the first unused byte in the last sector of a program file. No special symbols (such as >FF for the variable length file) are used to mark the end of the sector in a program file.

Assume the user are going to save a BASIC program that is 400 bytes long to disk. Sector bytes are numbered from 0 to 255 (a total of 256 bytes on the sector). The first sector will be completely filled. The file will occupy 144 bytes of the second sector. The remaining 112 bytes of the second sector will be unused. The pointer to the byte after the last used byte in sector two will be 144 (>90). The format of the sectors will be:



2.5.4 BASIC FORMAT DATA FILES. When a BASIC program accesses the disk, file management obtains a record from disk and passes the record to the BASIC interpreter. The interpreter must know the format of the data within the record. Currently BASIC supports display and internal data formats. Refer to the User's Reference Manual for additional information about data formats.

NOTE

The data format is a requirment of BASIC, not file management. The following rules must be

observed only for files that are to be accessed by BASIC programs.

2.5.4.1 Multiple Items in a Record. With BASIC it is possible to create records with more than one data item in each. When semi-colons (;) or commas (,) are used to separate variables in a PRINT statment, the variables are written to the same record. For example, the BASIC sentence:

```
PRINT #1:A$;B$
```

would first put A in the record. Next B would be placed in the record after A. Since B is not followed by a semi-colon the record would then be given to file management.

To read more than one item from a record a comma (,) is used. For example, the BASIC sentence:

```
INPUT #1:A,B
```

would take the first item from the record and place the item in the variable A. The second item would be removed from the same record and placed in variable B.

NOTE

If only one item of data is in the record, the BASIC interpreter will read the next record from the file to fill variable B.

2.5.4.2 Display Format. Display format is designed to act as if the data were coming from or going to the screen. Multiple data items in the same record must be separated by a comma. For example, the data record:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|D|I|S|P|L|A|Y| |D|A|T|A|,|2| |I|T|E|M|S|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

would contain two items. "DISPLAY DATA" is the first item in the record. The comma (,) marks the end of the first item. "2 ITEMS" is the second item in the record. The BASIC sentence required to create the record is:

```
PRINT #1:"DISPLAY DATA";",";"2 ITEMS"
```

NOTE

The user must manually place the comma (,) in the data. The system will not automatically insert a comma between data items.

If a comma is not in the data, the record consists of one item. For example:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
!D!I!S!P!L!A!Y! !D!A!T!A! !1! !I!T!E!M!
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

would be a record with only one item. Either of the following BASIC sentences would create the record:

```
PRINT #1:"DISPLAY DATA";" 1 ITEM"
PRINT #1:"DISPLAY DATA 1 ITEM"
```

Display format data can be in either fixed length or variable length records. The above examples are fixed length records (they do not contain a record length byte). An example of a variable length record with two items is:

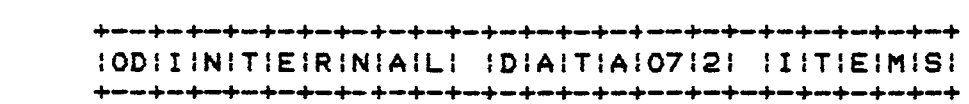
```
+--+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
!14!D!I!S!P!L!A!Y! !D!A!T!A!,!2! !I!T!E!M!S!
+--+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The >14 is one byte of hex data specifying the number of bytes in the record.

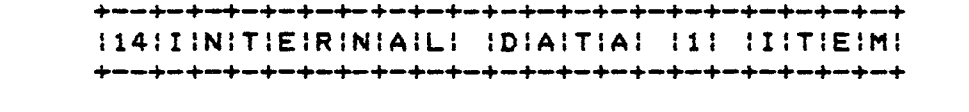
NOTE

Fixed length records are not always completely filled with data. For example, assume the word "TWO" is written to a fixed length record 80 bytes long. Only the first three bytes of the record would contain valid data. The remaining 77 bytes would be unused. Unused bytes of fixed length display format records must be filled with >20 characters.

2.5.4.3 Internal Format. With internal format files each item in the record has its own length indicator. Commas are not used to separate items within a record as in display format files. An example of a fixed length record, internal format with two items is:

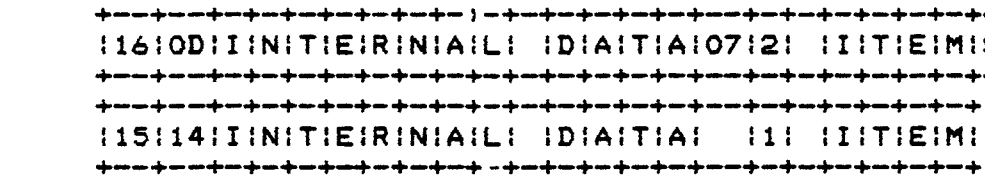


The >OD is one byte of hex data specifying there are decimal 13 bytes of data in the first item. The >07 is one byte of data specifying there are seven bytes of data in the second item. An example of a fixed length record, internal format with one item is:



The >14 is one byte of hex data specifying decimal 20 bytes of data in the first item.

The length indicators for internal format data should not be confused with the length indicators for variable length data records. One is used by the BASIC interpreter, the other by file management. Examples of variable length records with internal format data are:



The first byte of each record is the hex number specifying the amount of data in the record. This byte is used by file management.

NOTE

Fixed length records are not always completely filled with data. For example, assume the word "TWO" is written to a fixed length record 80 bytes long. Only the first three bytes of the record would contain valid data. The remaining 77 bytes would be unused. Unused bytes of fixed length

internal format records must be filled with >00 characters.

2.5.5 DISK DIRECTORY. Each diskette contains a disk directory that specifies the type and location of files currently on the disk. Each file on the disk has an entry in the disk directory. Eight bytes of the disk directory entry are important to the protocols. Bits are numbered 0 to 7 from left to right. The bytes are:

1. Byte 1-2 - Total number of sectors in the file.

2. Byte 3 - File flags.

   a. Bit 0 - Type of record. 0 = Fixed length records, 1 = Variable length records.

   b. Bits 1-3 - Not used.

   c. Bit 4 - Protect flag. 0 = Not delete/write protected, 1 = Delete/write protected.

   d. Bit 5 - Not used.

   e. Bit 6 - BASIC file information. 0 = Display format, 1 = Internal format. This bit is not used by file management.

   f. Bit 7 - Type of file. 0 = Data file, 1 = Program image.

3. Byte 4 - Number of records per disk sector. Not used with variable length or program image files.

4. Byte 5 - End of file offset. Not used with fixed length records. For variable length files points to the end of sector marker (>FF) for the last sector in the file. For example, if the EOF offset is >C0, at offset >C0 in the last sector of the file you would find the end of sector marker (>FF). For program image files points to the byte after the last data byte in the sector. For example, if the EOF offset is >C0, at offset >C0 in the last sector of the file would be the first unused byte of the sector. Offset >BF would contain the last byte of data in the sector.

5. Byte 6 - Record size. For fixed length records

contains the actual record length. For variable length records contains the maximum record size allowed. Not used for program image files.

6. Bytes 7-8 - Total number of records in the file with the order of the bytes reversed. Not used for program image files. For example, if a file contains 423 records (>01A7) the entry would be >A701.

2.5.6 END OF FILE DETECTION. Each of the three disk file types has its own method for detecting an end of file conditioin. The methods are:

1. Program Image - Bytes 1-2 of the disk directory specify the total number of disk sectors in the file. Maintain a count of the number of sectors processed. Continue processing until the count equals the number of sectors in the file. Remember, a program image file may use only part of the last disk sector. When processing the last sector use the end of file offset, specified in byte 5 of the disk directory, to determine the number of bytes actually used.

2. Fixed Length Records - Bytes 7-8 of the disk directory specify the total number of fixed length records in the file. Maintain a count of the number of records processed. Continue processing until the count equals the number of records in the file.

3. Variable Length Records - Bytes 1-2 of the disk directory specify the total number of disk sectors in the file. Maintain a count of the number of sectors processed. Continue processing until the count equals the number of sectors in the file.

SECTION 3

CODED DATA

3.1 REASONS FOR ENCODING DATA

The terminal emulator package cannot accept the full range of >0 to >FF as data in one byte. One bit of each byte is reserved for parity. This limits the maximum data range for a byte to >0 through >7F.

Some values are also reserved as control characters for communications systems. Transmission of data in the range >0 to >1F can have varying results from system to system.

The valid range for data transmission then is limited to >20 through >7F. Some of the special features of the emulator, such as sound, graphics and file transfer, require transmission of eight bit bytes. This means some data must be encoded so that transmitted bytes fall within the >20 to >7F range.

3.2 DATA CODING

Each coded byte of data contains six actual data bits. The bits are numbered 0 to 7 from left to right. Bit 0 is reserved for parity and can be ignored during encoding. Bit 1 is always set to 1 to ensure the byte will not be in the range >0 to >1F. Bits 2 to 7 contain the actual data.

Only six bits of actual data are allowed per coded byte. This method maps three bytes of normal data to four bytes of coded data. The mapping is:

	BYTE 1	BYTE 2	BYTE 3	BYTE 4
Normal Data	xxxx xxxx	yyyy yyyy	zzzz zzzz	
Coded Data	01xx xxxx	01xx yyyy	01yy yyzz	01zz zzzz

As an example, assume >1B >85 >F4 >01 is to be encoded. The binary representation of the four bytes is:

0001 1011 1000 0101 1111 0100 0000 0001.

The binary representation of the coded data is:

0100 0110 0111 1000 0101 0111 0111 0100 0100 0000 0101 0000.

The hex data after encoding is >46 >78 >57 >74 >40 >50.

NOTE

The emulator package does not validate bit 1 for a binary 1. If the host system has problems with transmitting a >7F the system can reset bit 1 to 0 converting the >7F to a >3F. Bit 1 should always be set to 1 if the transmitted byte would be in the range >00 to >1F with bit 1 set to 0. The emulator always sets bit 1 to 1 when transmitting encoded data to the host.

SECTION 4

EMULATOR CONTROL FORMATS

Control characters, escape sequences and extended writes are the three methods of passing control information to the emulator.

4.1 CONTROL CHARACTERS

Certain one byte codes in the range >0 to >1F have been reserved as control bytes. When the emulator is in the text mode and a control code is received, the emulator will perform a pre-defined function. Control codes supported are:

1. SOH (>01) - Home the cursor. When >01 is received the cursor is moved to the upper left hand corner of the display.

2. BEL (>07) - Produce a "beep" sound from the TI-99/4.

3. BS (>08) - Backspace the cursor one character position. Place a space (>20) in the position the cursor occupied before the backspace occured. As an example assume that "_" denotes the cursor and the current line is:

   ABCD_

   Two backspace bytes are received. The new line will be:

   AB_

4. LF (>0A) - Advance the cursor to the next line without changing the column position. For example, if the cursor is at column 12 line 4 and a >0A is received, the cursor will be positioned to column 12 line 5.

5. CR (>0D) - Return the cursor to the beginning of the current line. The cursor character will disappear from the display until another character is received.

NOTE

A carriage return (>0D) will not automatically generate a line feed (>0A).

6. FF (>0C) - Clear the screen and place the cursor in the upper left hand corner of the display.

NOTE

The above control characters are only in effect when the computer is in text mode. While in graphics mode codes >00, >0C and >0A will be ignored. The other codes will be displayed.

4.2 SPECIAL ESCAPE SEQUENCES

Escape sequences pass information to the emulator in both text and graphic modes. All escape sequences have >1B as their first byte. When >1B is received the emulator expects the next byte to be a special operation code defining the action to be performed. Supported escape sequence operation codes are:

1. >59 - Set the logical cursor address. The operation code followed by two bytes specifing the column and line number for the cursor. The position bytes are specified relative to >20. For example >1B >59 >30 >25 sets the cursor to column >10 (>30 - >20) and line >5 (>25 - >20).

2. >3A - Locks the keyboard on the computer. Keys pressed are not trasmitted until the keyboard is unlocked. The complete sequence is >1B >3A

3. >3B - Unlocks the keyboard on the computer. The complete sequence is >1B >3B.

4. >47 - Begin an extended write. A complete explanation of extended writes is given in section 4.3.

5. >48 - Home the cursor. Position the cursor in the

upper left hand corner of the display. This sequence will not clear the video display. The complete sequence is >1B >48.

6. >53 - System reset. Used to abort the file transfer. If file transfer is in progress the system exits the transfer and returns to normal processing. The complete sequence is >1B >53.

7. >79 - Place the computer in graphics mode. The command will clear the screen and place the logical cursor in the upper left hand corner of the display. A cursor character is not displayed in graphics mode. The complete sequence is >1B >79.

NOTE

The hardware requires about 90 milliseconds to switch modes. If any data is received during the switching period garbage will appear on the screen.

8. >7A - Place the computer in text mode. The command will clear the screen and place the cursor in the upper left hand corner of the display. The complete sequence is >1B >7A.

NOTE

The hardware requires about 90 milliseconds to switch modes. If any data is received during the switching period garbage will appear on the screen.

If the byte following the >1B is not one of the above codes the escape sequence is terminated and the received byte is ignored.

4.3 EXTENDED WRITES

The general format of the extended write is:

1B	47	7F	1B	28	OPER	DATA	1B	29

The >1B >47 >7F is the sequence which signals an extended write is about to begin. The >1B >28 indicates that the operation code is contained in the next byte.

OPER is a one byte extended write operation code specifiying the write being performed. DATA is the data associated with the extended write. The format of DATA depends on the write operation code. The sequence >1B >29 signals the end of the extended write.

Once an extended write has started (>1B >47 >7F >1B >28 has been received) the only way to terminate the write processing is with the sequence >1B >29. Any bytes between >1B >28 and >1B >29 are considered part of the write and will not be processed in the normal manner. Supported operations are listed below.

4.3.1 OPER >20 - DEFINE CHARACTERS. Command sequence to define one or more consecutive characters. The first two bytes specify the code of the first character to be defined. The remainder of the data is the information required to define the character(s) (see section 2.2.1.3). Valid character codes are >00 to >FF. The command requires the one byte character code be sent as two bytes. Bits 0-3 of the character code bytes are always 0010 (or >2). Bits 0-3 of the code byte are copied to bits 4-7 of the first character code byte. Bits 4-7 of the code byte are copied to bits 4-7 of the second character code byte. For example, character code >57 would be converted to >25 >27 in the command.

The data defining the character (all bytes between the character code and >1B >29) must be encoded as specified in section 3.

Assume the host wants to define character >8A as a T and character >8B as a L. Since the >8A and >8B are consecutive codes they can be defined with one extended write command. The command format calls for the code of the first character to be expressed as two bytes. In this example the two bytes would be >28 >2A.

Next the eight bytes defining each code must be determined. From the example in section 2.2.1.3 we know the eight bytes

required to define the T are >FF >FF >18 >18 >18 >18 >18 >18. Using the technique described in the example we find the data needed to define the L is >C0 >C0 >C0 >C0 >C0 >C0 >FF >FF.

The 16 bytes of data (eight bytes needed to define T and eight bytes needed to define L) must be encoded using the method described in section 3. The data after encoding is:

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|7F|7F|7C|58|46|41|60|58|46|41|63|40|70|4C|43|40|70|4C|43
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
+--+--+--+
|7F|7F|70|
+--+--+--+
```

The complete command sequence required to define character >8A as a T and >8B as an L would be:

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|1B|47|7F|1B|28|20|28|2A|7F|7F|7C|58|46|41|60|58|46|41|63
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|40|70|4C|43|40|70|4C|43|7F|7F|70|1B|29|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

4.3.2 OPER >21 - LOAD SOUND TABLES. Load a table with a sound list. Refer to section 2.3 for information on sound lists. The first byte is the sound table number + >20. The remainder of the data is the encoded sound list.

The emulator supports sound tables >0 to >F. Valid entries for the table number in the command are >20 to >2F. For example, to specify sound table >A the first byte would be >2A (>A + >20).

The sound tables are 128 bytes in length and contiguous in memory. Sound lists can be longer than 128 bytes. For example, assume a 200 byte sound list is to be loaded into table 0. Table 0 would contain 128 bytes of the list and table 1 would contain the remaining 72 bytes. Since part of table 1 is being used for the first sound list, the next list would have to be loaded into table 2.

4.3.3 OPER >22 - PLAY SOUND TABLE. Play the sound list in the sound table specified by the table number. The first byte is the table number + >20. For example, to specify sound table >A is to be played the first byte would be >2A (>A + >20). The complete command required to play a sound list in table >A is:

1B	47	7F	1B	28	22	2A	1B	29

4.3.4 OPER >23 - STOP SOUND. Stop playing sound lists. No information is required other than the operation code (>23). The command will turn off all sound generators. The complete command required to stop sound is:

1B	47	7F	1B	28	23	1B	29

4.3.5 OPER >24 - SELECT CHARACTER BANK. Select the character bank to be used for display. Refer to section 2.2.1.2 for details on character banks. The first byte specifies the bank to use. The code >47 (an ASCII L) selects the "lower" bank (character codes >0 to >7F) and >55 (an ASCII U) selects the "upper" bank (character codes >80 to >FF). The complete command required to select the upper bank is:

1B	47	7F	1B	28	24	55	1B	29

4.3.6 OPER >25 - DEFINE COLOR SETS. Define the foreground and background colors to be used for one or more consecutive color sets. Refer to section 2.2.1.4 for details on color sets. The first byte is the set number of the first color set to be defined + >20. The remaining data are the foreground/background color codes. The color code bytes must be encoded as specified in section 3.

Each byte after the set number specifies a foreground and a background color. Bits 0-3 of each byte specify the foreground color of a color set. Bits 4-7 of each byte specify the background color of a color set. For example, to specify a foreground color of black and a background color of cyan the byte would be >17. The 1 is the color code for black (2.2.1.4) and the 7 is the color code for cyan.

Assume color set >1A is to have a foreground color of black and a background color of cyan. Set >1B is to have a foreground color of dark red and a background color of light yellow. Set >1C is to have a foreground color of dark blue and a background color of gray. Since the three sets are consecutive, one extended write can be used to define the colors. The command

calls for the first byte to be the first set number to define + >20. In this example the byte would be >3A (>1A + >20). Using table 2-1 we find the color byte for the set >1A is >17, the color byte for set >1B is >6B and the byte for set >1C is >4E.

The bytes required to define the colors of the three sets are >17 >6B >4E. The three bytes must be encoded as described in section 3. The data after encoding is >45 >76 >6D >4E. The complete command required to define color sets >1A to >1C would be:

1B	47	7F	1B	28	25	3A	45	76	6D	4E	1B	29

4.3.7 OPER >26 - SPEAK AND DISPLAY TEXT. Display the text data contained within the extended write and speak the data after the write terminates. The text data must be upper case letters. For example, the complete command needed to have the emulator speak and display the phrase "HOW ARE YOU" is:

1B	47	7F	1B	28	26	48	4F	57	20	41	52	45	20	59	4F	55	1B	29

NOTE

The host must not transmit to the emulator while the emulator is speaking. Below are two methods which could be used to determine when speech has terminated.

1. The host should estimate the amount of time required to speak the string transmitted to the remote (the rate of speech will vary 10 to 15 per cent depending on the speech synthesizer). The host should pause for that estimated time interval.

2. The user on the remote should be instructed to press enter when speaking is completed. If the host receives enter (>0D) from the remote, the speech is completed and transmissions can continue.

4.3.8 OPER >27 - SPEAK TEXT WITHOUT DISPLAY. Speak the text data contained within the extended write. The only difference between this write and extended write >26 is that the text is not displayed to the screen using this write. Only upper case words can be spoken using this command. The complete command needed to have the emulator speak the phrase "HOW ARE YOU" is:

1B	47	7F	1B	28	26	48	4F	57	20	41	52	45	20	59	4F	55	1B	29

4.3.9 OPER >28 - SPEAK ALLOPHONES. Speak the allophone data contained within the extended write command. Refer to the manual supplied with the terminal emulator command module for a description of allophones. The allophone data must be encoded as described in section 3.

4.3.10 OPER >29 - LOOK UP WORDS. Equate the number specified in the command to the supplied text data. Find the text data in the speech dictionary. If the word is not in the dictionary the command will be ignored.

The first byte in the command is the number for the text data. The valid range for the byte is >20 to >7F. The rest of the data contains the word the number is to be associated with. The complete command required to associate the number >3A with the word "HELLO" is:

1B	47	7F	1B	28	29	3A	48	45	4C	4C	4F	1B	29

4.3.11 OPER >2A - SPEAK WORDS ASSOCIATED WITH NUMBERS. Each byte of data in the command is a number that has been equated to text using extended write >29. Speak the words associated with the numbers. Remember, extended write >29 must be used to associate the words with the numbers. Valid number ranges are >20 to >7F.

Assume extended write >29 was used to associate >20 with "HELLO", >35 with "HOW", >47 with "ARE" and >2A with "YOU". The complete command required to speak the sentence "HELLO HOW ARE YOU" is:

1B	47	7F	1B	28	2A	20	35	47	2A	1B	29

4.3.12 OPER >2B - CHANGE SCREEN COLOR. Change the screen color in graphics mode. This includes the border at the top and bottom of the screen. If the emulator is in text mode this write will set the foreground and background colors. Refer to section 2.2.2 for more information.

NOTE

The screen color is the color used when transparent is defined as a color to be used for characters. For example, assume character X has a foreground color of black and a background color of transparent. With the screen color set to cyan character X will be displayed as black and cyan. Changing the screen color to red displays character X as black and red.

The data consists of two bytes specifing the color. Bits 0-3 of both bytes contains 0010 (or >2). When in graphics mode bits 4-7 of the first byte contains the color of the screen. Table 2-1 gives the valid color codes. The second byte is ignored in graphics mode. For example, to set the screen color to cyan in graphics mode the complete command would be:

1B	47	7F	1B	28	2B	27	20	1B	29

In text mode bits 4-7 of the first byte contains the foreground color and bits 4-7 of the second byte contains the background color. For example, to set the foreground color to black and the background color to cyan in text mode the complete command would be:

1B	47	7F	1B	28	2B	21	27	1B	29

NOTE

Escape sequences specified in section 4.2 are used to switch between text and graphic modes.

4.3.13 OPER >2C - RETURN STATUS. Return the status of the terminal emulator module. The format of the reply is:

01	1B	28	VERSION	SCREEN	MODE	WRAP	1B	28

Each parameter requires one byte. The parameters are:

1. VERSION - Version of the emulator being used. The emulator version number is added to >20. For example, Terminal Emulator II would return >22 (>20 + >2).

2. SCREEN - Screen width selected by the user. Valid entries are 34, 36, 38 or 40 (>22, >24, >26 or >28).

3. MODE - Display mode currently in effect.

   a. >47 - Emulator is in graphics mode.

   b. >54 - Emulator is in text mode.

4. WRAP - Line wrap option currently in effect.

   a. >46 - Wrap is off. The emulator uses logical line lengths of 80 bytes to display the data.

   b. >4E - Wrap is on. The emulator uses logical line length selected by the user to display the data.

4.3.14 OPER >2D - TRANSMIT COMMAND. Initiates the file transfer. The format of the command is explained in section 5.

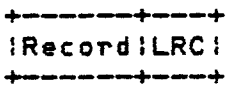
SECTION 5

FILE TRANSFER

The terminal emulator supports the transfer of files between computer systems. The protocol provides for full error detection during the transfer.

5.1 LRC ERROR DETECTION

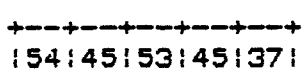
The emulator uses the Longitudinal Redundancy Check (LRC) method of error detection. During the file transfer the sender calculates an LRC byte for each record transmitted. The LRC byte is transmitted as the last byte of the record. The LRC must be greater than >21. If the calculated LRC is less than >21 then >21 is added to the value before transmission. The format of the transmission is:



The LRC is calculated by performing successive exclusive_or operations (XOR) on all the bytes in the record. If the result is less than >21 then >21 is added to the LRC. As an example assume that the record to be transmitted is 4 bytes long and consists of >54 >45 >53 >54. To calculate the LRC we would first XOR >54 and >45.

```
          >54      0101 0100
          >45      0100 0101
          ------------------
Result - >11       0001 0001
```

The result of >54 XOR >45 is >11. Next we XOR >11 and >53 giving >42. Finally we XOR >42 and >54 giving >16. Since the XOR of the record is less than >21 we must add >21. The LRC byte is >37 (>16 + >21). The transmission for our example would be:



5.2 TRANSMISSION BLOCKS AND RECORDS

The protocol divides the file being transferred into blocks. The blocks are further sub-divided into records. The size of the blocks and records depend on the file being transmitted. Table 5-1 gives tha various record sizes and number of records per block for the different files.

Table 5-1 TRANSMISSION RECORD SIZE

Device	Data Encoded for Transfer	Logical Record Size	Transfer Record Size	Transfer Records/Block
DISK	No	1-256	64	4
DISK	Yes	1-256	69	5
CASSETTE	No	1-128	64	4
CASSETTE	No	129-192	64	3
CASSETTE	Yes	1-128	57	6
CASSETTE	Yes	129-192	64	4

As an example of the use of the table, assume we are transfering a disk file. The data does not need to be encoded (all bytes in the file are in the range >20 to >7F) and the logical record size is 95. For column one we have "disk", for column two we have "no" and column three does not affect disk files. Thus we use row one to get the block information. For this example each transmission record will contain 64 bytes of data and there will be four transmission records in each block.

Assume we are going to transfer a file from cassette. The data does need to be encoded (some bytes are in the range >00 to >1F or >80 to >FF) and the logical record size is 100. For column one we have "cassette", for column two we have "yes" and for column three we have 1-128. Thus, we use row five to get the block information. For this example, each transmission record would contain 57 bytes of data and there would be six transmission records in each block.

Each block has a block number assoicated with it. The block number is a two byte value that starts at >2020. The first block in the file has a number of >2020, the second block has a number of >2021 and so on. The maximum value allowed in a block number byte is >7E. Block >7E in the file has a block number of >207E. Block >7F in the file however has a block number of >2120.

Each block is divided into records. The number of records in a block is given in Table 5-1. The record number is a one byte value that starts at >20. The record number resets to >20

for each data block transmitted.

5.3 TRANSMIT COMMAND FORMAT

Extended write >2D is the transmit command. It is used to notify the receiving system that a file transfer is to occur and to pass parameters to the receiver. The format of the command is:

1B	47	7F	1B	28	2D	DATA	1B	29	LRC

The DATA part of the transmit command must be encoded as described in section 3. The format of the data is:

1. Byte 1 - Denotes the type of file being transfered. Valid codes are:

   a. >43 - File is coming from cassette.

   b. >44 - File is coming from disk.

2. Bytes 2-9 - Disk directory information. Bytes 2-6 and 8-9 only have meaning if byte 1 contains a >44. Byte 7 always has meaning. Refer to section 2.5.5 for a complete description of the eight bytes. Remember, the disk directory information in the transmit command starts at byte 2. The format of the eight bytes is:

   a. Bytes 2-3 - Number of sectors in the file.

   b. Byte 4 - File flags. Refer to section 2.5.5 for more information about the flags.

   c. Byte 5 - Number of records per sector. Not used for fixed length records or program image files.

   d. Byte 6 - End of file offset.

   e. Byte 7 - Record size. Byte 7 is required for both disk and cassette data files.

   f. Byte 8-9 - Number of records in file, with bytes reversed. Not used for program image files.

3. Bytes 10-11 - Data record size. Length of DATA in transmission record.

4. Byte 12 - Number of transmitted records per block.

5. Byte 13 - Emulator version number. Terminal Emulator II is version number >01.

6. Byte 14 - Delay interval in increments of 5 seconds. This is the number of 5 second intervals the receiver is to wait for transmission from the sender before assuming data loss has occured. For example, if this byte contains 4 the receiver is to wait 20 seconds (4 times 5) for transmission from the sender. If no transmission is received a NAK record is to be sent to the host.

7. Byte 15 - Type of error checking. Current versions only support LRC error checking. Byte 15 must be set to >4C for LRC checking.

8. Byte 16 - Type of data coding. Valid entries are:

   a. >36 - Coded file data. Data is encoded as described in section 3.

   b. >37 - ASCII file data. No encoding is done. Data must be in the range >20 to >7F or unpredictable results will occur!

9. Byte 17 - Type of data record retransmission to use. Current versions will only support >4E as a valid entry.

5.4 TRANSMISSION RECORD FORMATS

There are three transmission record formats: one for the first record, one for all records except the first and last records and one for the last record. The format of the first data record is:

The bytes >02 and >01 signal the start of a data record. Byte >1D signals that the two following bytes will contain the block number. BLK is a two byte area specifing the block number of the record being transmitted. Section 5.2 describes the block number.

Byte >1E signals that the next byte will be the record number. REC is a one byte area specifing the record number of the record being transmitted. Section 5.2 describes the record number. Remember, the number of records in a block is specified in the transmit command.

Bytes >1B and >28 signal the beginning of the data portion of the first record. DATA contains the file data. The length of the transmitted DATA (not the length of data before encoding) is specified in the transmit command. The transmit command also tells if the data has been coded or not. If the transmit command specifies encoding, only DATA will be coded, not the other bytes in the transmission.

NOTE

If the data is coded it is coded as a block, not as a record. To decode the data correctly, the entire block must be decoded.

A >17 specifies the end of the data record. The LRC byte always follows the >17 code.

The data record format for all records between the first and last records is:

The only difference between this format and the first one is that this format does not contain the >1B >28 before DATA.

The data record format for the last record in the file is:

The only difference between the previous format and this one is this format ends in an >1B >29 >03. These three bytes tell the remote system that this is the last record in the file transfer.

5.5 ACK RECORD FORMAT

An ACK record is sent for each data record correctly received. The format of the record is:

The BLK field is a two byte field containing the block number of the received record. The REC field is a one byte field containing the record number of the received record. A >06 is the ASCII ACK code.

5.6 NAK RECORD FORMAT

A NAK record is sent whenever an error is detected in a transmission. The format of the record is:

BLK is a two byte area containing the block number the sender of the NAK was expecting. REC is a one byte area containing the record number the sender was expecting. A >15 is the ASCII NAK code. The ID field is one byte that specifies the type of record being NAK'D. A >30 indicates the NAK is for a data record, a >31 indicates the NAK is for a response record (a NAK or ACK record).

The sender should maintain a count of the number of NAKs sent while trying to transfer a record. If more than five NAKs are encountered for any single record the sender should abort the transfer. When the emulator is sending the file it will allow up to five NAKs. When the emulator is receiving the file it will allow up to eight NAKs.

5.7 FILE TRANSFER FROM REMOTE TO HOST

Appendix B provides an implementation example of remote to host file transfer.

The protocols use the transmit command (extended write >2D) to signal a file transfer is about to occur and to pass parameters to the host. Section 5.3 describes the transmit command. Hosts that echo the bytes they receive must know in

advance that the file transfer is about to start. The echo must be turned off before the transmit command is received.

The host calculates an LRC byte for the write (as described in 5.1) and compares the calculated value to the received LRC value. If they do not match the host sends a system reset (>1B >53) terminating the transfer.

Once the host determines the transmit command was received without error a read buffer command (>1B >38) is transmitted to the remote. The read buffer informs the remote that the host has received and processed the transmit command and is ready to start the file transfer. The remote then begins sending transmission records to the host. Section 5.4 describes the format of data records.

The error checking is the same for all received transmissions (data records and ACK/NAK responses). First the LRC of the record must be calculated. If the calculated LRC does not match the received value a NAK is sent to the remote using the expected block and record numbers. The host then waits for the remote to resend the record.

Provided the LRC bytes match, the host checks the block number of the record against the expected block number. If the block number does not match the expected value a system reset is sent (>1B >53). The host exits the file transfer and returns to normal processing.

Next the host checks the expected record number against the received record number. If the numbers do not match the host sends a NAK using the expected block and record numbers. The host then waits for the remote to retransmit the record.

If no errors are detected the host processes the data record. The expected record number is incremented to the next data record. If this is the last record in a block, the block number is incremented to the next block and the record number is set to >20. An ACK is sent for the record just received and the host waits for the next record.

After the last record is received without errors and the host sends an ACK, the remote will send an end of file record. The EOF record is an ACK with a block number of >7E7E and a record number of >7E. The EOF ACKs will be refered to by number. The EOF ACK the remote sends to the host after the last record is ACK-1.

The host does normal error checking on ACK-1. If no errors are detected the host sends the same ACK (ACK-2) back to the

remote. If the remote does not receive ACK-2 correctly it will send a NAK back to the host. If the remote does receive ACK-2 it will send the same ACK (ACK-3) back to the host.

The host is now awaiting ACK-3 from the remote. Four things can happen.

1. The host can receive a NAK from the remote. If the host does receive a NAK it will resend ACK-2 and continue to wait for ACK-3.

2. The host can receive ACK-3 from the remote. If the ACK is received the host is to exit file transfer and return to normal processing.

3. The host can receive a transmission from the remote with an error. If an error occurs the host is to exit the file transfer and return to normal processing.

4. The host can time out waiting for a response. If the host does time out it is to exit the file transfer and return to normal processing.

While reading bytes from the remote the host constantly checks for a system reset (>1B >53). If the reset is detected the host terminates the file transfer and returns to normal processing.

The transmit command specifies a time interval to the host. The interval is the maximum time to wait for a byte from the remote. If the time limit is exceeded without receiving a byte the host assumes the transmission was lost and sends a NAK using the expected block and record numbers.

The transmit command specifies how the data is sent. Files with variable length records or files with data in the range >0 to >1F or >80 to >FF are encoded as described in section 3. If the host is to manipulate the file it must be decoded. The remote encodes a block of data at a time. The host must decode the data a block at a time. It cannot decode the records independently of one other.

5.8 FILE TRANSFER FROM HOST TO REMOTE

Transmissions received by the host cannot be echoed. Echo must be turned off before the file transfer can begin. Appendix C provides an implementation example of host-to-remote file

transfer.

When transferring a file to a TI-99/4 disk file the host must construct a 256 byte disk sector. The sector is transmitted as one transmission block. Transfer to cassette does not require a knowledge of how the file is stored on the cassette. The records are moved directly into the transmission block.

The host must construct and send the transmit command (extended write >2D) to the remote. The extended write is used to notify the remote a file transfer is to occur and to pass parameters to the remote. Section 5.3 describes the information contained in the write. The host calculates an LRC byte for the transmit command and attaches the LRC to the end of the command.

After the extended write is transmitted to the remote the host waits for the read buffer command (>1B >38). The read buffer notifies the host that the remote has processed the extended write and is ready to receive the file transfer. The host does not time out waiting for the read buffer.

After the read buffer is received, the host gets a block of data. If the transmit command specified the data is to be encoded, the host encodes the entire block using the method described in section 3. Any file with data in the range >0 to >1F or >80 to >FF must be encoded. Also files with variable length records must be encoded.

The host divides the block into data records. The length of the data record is specified in the transmit command. The host transmits the block a record at a time. An LRC byte is calculated for each record and attached as the last byte of the record. After the record is transmitted the host waits for a reply from the remote. If the host waits longer than the time specified in the transmit commmand the host assumes the response has been lost and sends a NAK to the remote using the block and record number of the transmitted record.

The host performs extensive error checking on the responses. First the host calculates an LRC byte for the response. If the calculated LRC does not match the received LRC the host sends a NAK using the block and record number of the last record transmitted. If the LRC is correct the host compares the received block number to the block number of the last record transmitted. If the block numbers do not match the host sends a system reset (>1B >53) and exits the file transfer. If the block numbers match the host compares the received record number to the record number of the last record transmitted. If the record numbers do not match the host sends a NAK using the block and record numbers of the last record transmitted.

If there are no errors in the response the host checks for an ACK or a NAK response. If the remote sent a NAK the host retransmits the last record and then waits for a response. If the remote sent an ACK the host transmits the next record.

After the host has processed the response for the last record in the transfer it sends an end of file record. The EOF record is an ACK with a block number of >7E7E and a record number of >7E. The ACKs will be refered to by number. The EOF ACK is ACK-1.

After sending ACK-1 the host waits for a response. When the response is received it goes through the normal error checking. If an error is detected a NAK is sent with block and record numbers of >7E. If no errors are detected and the response is a NAK the host retransmits ACK-1 and waits for a response. If the response is an ACK (ACK-2) the host transmits an ACK (ACK-3) using block and record numbers of >7E and exits the file transfer.

The host constantly checks the input for a system reset (>1B >53). If a system reset is detected the host exits the file transfer and returns to normal processing.

The host specifies a time interval in the transmit command. The interval is the maximum time the remote is to wait for a byte from the host. The host should use the same interval as the maximum time it will wait for a byte from the remote. If the time limit is exceeded without receiving a byte the host assumes the transmission was lost. It sends a NAK using the block and record numbers of the last data record transmitted.

APPENDIX A

DEFAULT CHARACTER SET

The terminal emulator supplies a default character set. When the emulator switches to text mode the default character set is loaded.

Hex code	Character	Hex Code	Character
20	Space	30	0
21	!	31	1
22	"	32	2
23	#	33	3
24	$	34	4
25	%	35	5
26	&	36	6
27	'	37	7
28	(	38	8
29	)	39	9
2A	*	3A	:
2B	+	3B	;
2C	,	3C	<
2D	-	3D	=
2E	.	3E	>
2F	/	3F	?

Hex code	Character	Hex Code	Character
40	@	60	`
41	A	61	a
42	B	62	b
43	C	63	c
44	D	64	d
45	E	65	e
46	F	66	f
47	G	67	g
48	H	68	h
49	I	69	i
4A	J	6A	j
4B	K	6B	k
4C	L	6C	l
4D	M	6D	m
4E	N	6E	n
4F	O	6F	o
50	P	70	p
52	R	72	r
53	S	73	s
54	T	74	t
55	U	75	u
56	V	76	v
57	W	77	w
58	X	78	x
59	Y	79	y
5A	Z	7A	z
5B	[	7B	{
5C	\	7C	|
5D	]	7D	}
5E	^	7E	~
5F	_		

APPENDIX B

EXAMPLE OF EMULATOR TO HOST FILE TRANSFER

This appendix lists one method for implementing the emulator to host file transfer on the host. It is intended only as an example of the file transfer on the host.

1. Turn off echo of incoming characters.

2. Wait for transmit command.

3. Transmit command received.

   a. Check LRC byte for error. If error send reset command and exit file transfer.

   b. No error, set expected block number to >2020 and expected record number to >20. Send read buffer command.

4. Wait for response.

   a. Start timer.

   b. If timer expires send NAK using expected block and record numbers. Go to step 4.

   c. If system reset received exit file tranfer.

5. Response received.

6. Check LRC byte for error. If error send NAK using expected block and record number. Go to step 4.

7. Check received block number against expected block number. If not equal send system reset. Exit the file transfer.

8. NAK Received. Resend the response for the received record number. Go to step 4.

9. Record received.

   a. Check received record number against expected record number. If not equal send NAK using

expected block and record numbers. Go to step 4.

b. Extract data from record and save into accumulation area.

10. Send ACK using expected block and record numbers.

11. Increment the expected record number. If this is the last record in the block:

    a. If data is encoded then decode the data.

    b. Save data to disk.

    c. Increment expected block number. Set record number to >20.

12. If not the last record in the transmission go to step 4.

13. End of transmission logic. Wait for response.

    a. Start timer.

    b. If timer expires send NAK using block number of >7E7E and record number of >7E.

    c. If system reset received exit the file transfer.

14. Response received.

15. Check LRC byte for error. If error send NAK using block number of >7E7E and record number of >7E. Go to step 13.

16. NAK received. Resend the ACK for the last data transmission. Go to step 13.

17. ACK received.

    a. Check received block number for >7E7E. If not equal send system reset. Exit the file transfer.

    b. Check received record number for >7E. If not equal send NAK using block number of >7E7E and record number of >7E. Go to step 13.

18. Send ACK using block number of >7E7E and record number >7E.

19. Wait for response.

    a. Start timer.

    b. If timer expires exit the file transfer.

20. Response received.

    a. If response has an error in it exit the file transfer.

    b. If response is a NAK send ACK using block number of >7E7E and record number of >7E. Go to step 19.

21. Exit the file transfer.

APPENDIX C

EXAMPLE OF HOST TO EMULATOR FILE TRANSFER

This appendix contains one method for implementing the host to emulator file transfer on the host. It is intended only as an example of the file transfer on the host.

1. Turn off echo of incoming characters.
2. Construct transmit command.
3. Send transmit command to remote.
4. Wait for responses.
   a. If system reset received exit the file transfer.
   b. If anything other than a read buffer command is received send a system reset and exit the file transfer.
5. Set current block number to >2020.
6. Read a block from disk. If end of file encountered go to step 17.
7. Set current record number to >20.
8. If data needs to be encoded then encode the block
9. Break the block into transmission records.
10. Send a transmission record.
11. Wait for response.
    a. Start timer.
    b. If timer expires then send a NAK using the current block and record numbers. Go to step 11.
    c. If system reset received exit the file transfer.
12. Response received.

13. Check LRC of the response. If error send NAK using the current block and record numbers. Go to step 11.

14. Check received block number against the current block number. If they are not equal send a system reset and exit the file transfer.

15. NAK received. Go to step 10 (this resends the record).

16. ACK received.

    a. Increment current record number.

    b. If all records in current block transmitted increment current block number. Go to step 6.

    c. Go to step 10 (this sends next record in the block).

17. End of transmission logic. Send ACK with block number of >7E7E and record number of >7E.

18. Wait for response.

    a. Start timer.

    b. If timer expires then send NAK with block number of >7E7E and record number of >7E. Go to step 18.

    c. If sytem reset received exit the file transfer.

19. Response received.

20. Check LRC of the response. If error send NAK using a block number of >7E7E and a record number of >7E. Go to step 18.

21. Check the received block number against >7E7E. If they are not equal send a system reset and exit the file transfer.

22. Check the received record number against >7E. If they are not equal send a NAK using a block number of >7E7E and a record number of >7E. Go to step 18.

23. NAK received. Go to step 17 (this resends the ACK).

24. ACK received.

a. Send an ACK using a block number of >7E7E and a record number of >7E.

b. Exit the file transfer.