
Part III/ Appendices

Appendix	Page
A/ Job Control Language	A-3
B/ Model 4 Hardware	A-35
C/ Character Codes	A-45
D/ Error Messages and Problems	A-61
E/ Converting TRSDOS Version 1 BASIC Programs to TRSDOS Version 6 BASIC Programs.....	A-75
F/ BASIC Keywords and Derived Functions	A-79
G/ BASIC Worksheets	A-83
H/ Glossary	A-85
I/ TRSDOS Programs.....	A-91
J/ BASIC Memory Map.....	A-101
K/ Using The Device-Related Commands	A-103
L/ HERZ50.....	A-107
M/ Backup Limited Diskettes.....	A-109

Appendix A/ Job Control Language

The TRSDOS Job Control Language (JCL) is one of the most powerful features of TRSDOS. It consists of:

- TRSDOS commands
- Macros
- Special symbols

You can use JCL to make your computer more “user friendly.” That is, you can write JCL programs that perform a variety of functions, such as FORMAT and BACKUP, and have TRSDOS execute these functions when the user types in one command line.

If you have read the entries on the BUILD and DO commands, you know how to create a JCL file composed of TRSDOS commands. You can make this file more powerful by utilizing macros and other features of JCL. This section describes how.

The steps for creating and using a JCL file are:

1. Create a JCL file consisting of TRSDOS commands, macros, or special symbols. You can do this with the BUILD command, SCRIPSIT, or a BASIC program.
2. Execute the JCL file with the DO command. This causes the JCL processor to:
 - Take control of the keyboard (for line input)
 - Read a line in the DO file exactly as if it came from the keyboard
 - Return control of the keyboard to the user when it reaches the last line.

The following sections give complete information on all the JCL features:

- Simple JCL Execution
- Simple JCL Compiling
- Advanced JCL Compiling

Simple JCL Execution

This section lists the execution macros and gives examples on how to create and run a JCL file.

Creating a JCL File

A JCL file contains characters normally available from the keyboard (ASCII characters).

There are several ways to create a JCL file: the BUILD library command lets you create or extend a JCL file, but it does not let you edit an existing file. You can create and edit a JCL file with a BASIC program. A word processing system, such as SCRIPSIT, will also let you create or edit a JCL file.

Restrictions of JCL

- A JCL file line cannot be longer than 79 characters. Depending on the JCL method used (execute only or compile), JCL either ignores all characters after the 79th or aborts the processing entirely.
- Any program or utility with unpredictable prompts will not function properly when run from a JCL file.
- Any program or utility which requires removing the system disk causes the JCL to abort.
- You cannot execute certain TRSDOS library commands and utilities from a JCL file. The commands and utilities NOT valid from a JCL file are : certain BACKUP commands, BUILD, certain CONV commands, all (X) commands, DEBUG, certain PURGE commands, SYSGEN, and SYSTEM (SYSTEM=) command.
- As a general rule, you should not use a library command or utility when you specify the QUERY parameter.

Table 1/ Execution Macros

Macro Group	Group Description	Macros	Macro Description
Execution Comment		.Comment	Displays a comment on the screen during execution. Comments are written to SYSTEM/JCL.
Termination Macros	Terminate execution.	//ABORT //EXIT //STOP	Stops execution, displays "Job aborted". Returns to TRSDOS or BASIC Ready. Stops execution, displays "Job done". Returns to TRSDOS or BASIC Ready. Stops execution. Returns control to the user program.
Pause/Delay Macros	Provide special functions.	//PAUSE //DELAY //WAIT //SLEEP	Suspends execution and displays a message. Suspends execution and displays a message for a specified amount of time. Suspends execution depending upon the setting of the system clock. Suspends execution for a predetermined amount of time.
Alert Macros	Provide video and audio alerts.	//FLASH //ALERT	Flashes a message on the screen a specified number of times. Provides an audible signal to the operator.
Keyboard Macros	Accept keyboard input.	//KEYIN //INPUT	Selects predefined blocks of JCL lines. Inputs a line of information from the keyboard.

JCL Execution Macros

A macro is a pre-defined JCL instruction. `//ABORT` is an example of a macro symbol. Macro symbols must start at the first character position in the line. **An execution macro cannot be the first line in a JCL file.**

The JCL execution macros are:

`//ABORT`

Use this macro to exit a JCL procedure (if an error is encountered) and return to the program that initiated the `DO` command.

Your system returns you to the calling program if your JCL processing logic detects an error. The following message:

```
Job aborted
```

is displayed when an error is encountered.

`//ALERT [(tone,silence,tone,silence, . . .)]`

Use this macro to produce tones to the operator. `//ALERT` can generate up to eight different tones using the sound generator inside the computer.

You could use this macro to signify the end of a large JCL procedure. It could also be used during the execution of a procedure to bring attention to a specific process.

Tone is controlled by a number ranging from 0 - 7, with 7 producing the lowest tone and 0 producing the highest tone.

The tone is followed by a period of silence which you select with a second number ranging from 0 - 7, with 7 producing the longest period of silence and 0 producing the shortest period of silence. Tone and silence must be entered as number pairs (for example, "1,0"). You can enter as many number pairs as can fit on one line.

You can repeat the tone-silence sequence by enclosing the entire string in parentheses. The sequence keeps repeating until you press **ENTER**, which continues execution of the JCL. Pressing **BREAK** aborts the JCL.

Any value entered (for tone or silence) is used in its modulo 8 form. That is, if you enter the number 8, a zero value is assumed. For example, the value 10 produces the tone assigned to 2.

`//DELAY duration`

The `//DELAY` macro provides a definite timed pause with execution automatically continuing at the end of the delay. The actual delay will be approximately 0.1 second per count. The count ranges from 1 to 256. Thus, a delay of from 0.1 second to 25.6 seconds is possible.

You could use the //DELAY macro to suspend execution long enough for you to make sure the printer is ready to print.

The execution time of a //DELAY macro will vary slightly according to the speed TRSDOS is running under (FAST or SLOW). See the SYSTEM library command.

//EXIT

Use this macro to halt execution of JCL processing and return to the program that initiated the DO command.

If you do not enter a termination macro in a JCL file, the JCL processing terminates when it reaches the end of the file (as if //EXIT were the last line in the JCL file). The following message is displayed:

```
Job done
```

This message indicates a normal conclusion of the JCL file.

You should use //EXIT if the conclusion of the JCL file also represents the conclusion of the job that is running. So, //EXIT can be used to conclude a program that does not require any more keyboard input, and needs to return to TRSDOS Ready or BASIC Ready after it finishes.

To conclude a program that requires additional keyboard input, use the //STOP macro. Using the //EXIT macro would terminate the program.

//FLASH [*duration*] *message*

This macro flashes *message* on and off the video screen. *duration* is the number of times the *message* will flash and can be any number from 0 to 255. If *duration* is not specified, the message flashes 256 times. The message is any comment that you want displayed (up to 72 characters).

//KEYIN [*comment string*]

Use this macro to prompt for a single character entry (0 - 9), with the entire //KEYIN line being displayed.

During execution, press the appropriate character (0 - 9) to select the corresponding execution block in a JCL file. There can be up to ten execution blocks in a JCL file, each tagged with // and a number 0 - 9.

Do not use //KEYIN to enter data at execution time. If you do need to enter data at execution time, use the //INPUT macro.

//INPUT [*message string*]

Use this macro to input a line from the keyboard during JCL execution. With this macro, control of the keyboard is temporarily

returned to the operator. Now, any command can be typed on the keyboard and then passes to the system.

The number of characters allowed in the input line depends on where the JCL execution is when the //INPUT is encountered. For example, if the JCL is executing at the TRSDOS Ready level, then you can enter up to 80 characters, the same as for a normal TRSDOS command. If the //INPUT is encountered after going into BASIC, then you can enter up to 255 characters.

When you use the //INPUT macro, you should exercise some caution to assure that the command typed in is valid at the level it will be executed. For example, if you enter a program name incorrectly, the error message "Program not found" is displayed and the JCL execution aborts.

//PAUSE [*message string*]

When this macro is encountered in an executing JCL file, it is displayed on the screen along with a message. You can use the message to inform the operator why the pause was ordered. Press **(ENTER)** to resume JCL execution, or press **(BREAK)** to abort the JCL.

The //DELAY, //WAIT, and //SLEEP macros are similar to the //PAUSE macro, and are used to give JCL execution a specific delay period.

//SLEEP *hh:mm:ss*

Use this macro to put the system "to sleep" for the amount of time you specify.

//SLEEP adds the specified time to the current system time and waits until that time to begin or resume execution.

Suppose you have two programs that begin execution every morning at 10 o'clock and each program runs for two hours. You could execute the first program and have the //SLEEP macro "halt" execution of the second program for an hour lunch break. After the system "sleeps" for the specified hour, the second program is executed.

//STOP

Use this macro to halt execution of a JCL file and return keyboard control to the applications program that requests additional keyboard input.

If you do not use the //STOP macro, you automatically return to TRSDOS Ready or BASIC Ready. You can also use the //ABORT and //EXIT macros to force an end to the JCL execution and return to TRSDOS Ready or BASIC Ready.

//WAIT hh:mm:ss

The //WAIT macro is similar to //DELAY, except that the length of the delay depends on the setting of the system clock.

The //WAIT macro puts the entire system in a "sleep" state until the system clock matches the time you specified.

You can set the system clock with the TIME library command. You can also set the time from a JCL file by using a direct execution of the TIME library command, or with the //INPUT macro. Set the clock in the format hh:mm:ss.

Examples

The easiest JCL file to understand is one containing only commands.

Use the BUILD command to create the following JCL file named START/JCL:

```
DEVICE
FREE
```

If you issue a DO = START command (see the DO library command), your computer displays the device table, lists free space information about all enabled drives, and returns to TRSDOS Ready.

Because an execution macro cannot be the first line in a JCL file, you could use an execution comment to display an informative message as the JCL file begins to execute. An execution comment begins with a period, which must be in the first character position of the line. You could label START/JCL as follows:

```
. This program executes the DEVICE and FREE
commands.
DEVICE
FREE
//EXIT
```

This comment describing the file's purpose is displayed when the JCL executes. It shows the file's purpose. Notice that we added the termination macro //EXIT.

You can use the //PAUSE macro in START/JCL as follows:

```
. This program executes the DEVICE and FREE
library commands.
//PAUSE Be sure the correct disk is in Drive 0!
DEVICE
FREE
//EXIT
```

This example suspends the JCL before DEVICE executes, so you can be sure that the correct disk is in Drive 0. Press **(ENTER)** to continue the JCL.

You can use the //DELAY macro if you want to display an informative message to the operator. For example:

```
. BE SURE THAT THE PRINTER IS TURNED ON!  
//DELAY 50  
DEVICE (P)  
FREE (P)  
//EXIT
```

This example displays the above informative message and delays execution for approximately 5 seconds. After the delay, it executes DEVICE and FREE.

If you want your system to execute START/JCL at a certain time of the day, use the //WAIT macro as follows:

```
. This program runs at 2:15 a.m.  
//WAIT 02:15:00  
DEVICE  
FREE  
//EXIT
```

This example displays the comment and then waits until the system clock matches the time of 02:15:00 specified in the //WAIT macro. It would then execute DEVICE and FREE, and return to TRSDOS Ready.

```
. This program runs after a two-hour pause.  
//SLEEP 02:00:00  
DEVICE  
FREE  
//EXIT
```

This example displays the comment and then “sleeps” for two hours. It then executes DEVICE and FREE, and returns to TRSDOS Ready.

To use the //FLASH macro, modify START/JCL as follows:

```
. This program executes the DEVICE and FREE  
commands.  
DEVICE  
//FLASH 10 Starting execution of FREE  
FREE  
//EXIT
```

After DEVICE executes, the //FLASH line is displayed. It flashes on and off 10 times, as specified by the duration count. You can press **(ENTER)** to stop the flash and proceed to the next line. Pressing **(BREAK)** while the message is flashing aborts the JCL and displays the message “Job Aborted”.

You can modify START/JCL to show several uses of //ALERT:

```
. This program shows several uses of //ALERT.  
. TURN TO PAGE 4 AT THE TONE.  
//ALERT 0,0,1,5,0,2  
. PRESS ENTER TO BEGIN EXECUTION.  
//ALERT (1,0,7,0)  
DEVICE  
FREE  

```

The first tone tells you when to turn to page 4. The second tone repeats until you press **(ENTER)** to continue execution of the program or **(BREAK)** to abort the JCL.

The next example shows how you could build a menu using execution comments to display different program choices. Using the //KEYIN macro lets you press a single key to execute the desired program.

```
. START/JCL  
. Program 1 is FREE :0  
. Program 2 is FREE  
. Program 3 is DEVICE  
//KEYIN Select program, 1 - 3  

```

There are two new macros used in this example. They are //number and ///.

//number is used to start a block of lines that corresponds to a value selected with the //KEYIN macro. This block extends until the next //number or to the ///.

/// (the triple slash) is used to mark the end of all //number blocks. JCL stops looking for a match as soon as it encounters a ///. Execution continues with the following line.

In the above example, pressing 1, 2, or 3 selects the corresponding block of lines and runs the appropriate command. If you press a key other than 1, 2, or 3, all three //number blocks are ignored, and execution continues with the line after the ///.

The lines following the /// could contain other command options or an //ABORT macro to abort the JCL. One possible option could be to let the operator type in his own command.

Consider the following rewrite of START/JCL that uses the //INPUT macro to let the operator type in his own command:

```
. START/JCL
. Program 1 is FREE :0
. Program 2 is FREE
. Program 3 is DEVICE
//KEYIN Select program, 1 - 3
//1
FREE :0
//EXIT
//2
FREE
//EXIT
//3
DEVICE
//EXIT
///
//INPUT Enter your own choice of command.
//EXIT
```

Now, if you press a key other than 1, 2, or 3 for the //KEYIN, the //INPUT line is displayed.

You can also enter information directly into the system at the JCL level. For example, the //WAIT macro description mentions that you can set the time for the system clock in the middle of a JCL file.

The following example prompts you to enter the TIME library command to set the system clock. After you input the time, the //WAIT macro pauses execution of the JCL file until the clock matches 02:15:00 and then continues execution.

```
. This program runs at 2:15 a.m.
//INPUT Enter the TIME command using HH:MM:SS format.
//WAIT 02:15:00
DEVICE
FREE
//EXIT
```

JCL Compiling

The previous section explained how to create and use execute JCL files. This section describes some basic functions of the JCL compiler and shows practical examples of JCL files.

While an execute JCL file is useful, you need to use the compile phase of JCL for extra features. These extra features are explained in four parts:

- Compilation Description and Terms
- Conditional Decisions
- Substitution Fields
- Combining Files

Compilation Description and Terms

You can compile and/or execute any JCL file using the DO library command. If your JCL file contains only execution comments, commands, or execution macros, then you can completely skip the compile phase (using the "=" control character with the DO command).

If, however, your JCL file contains "tokens" or labels, or must make logical decisions, then you must compile the file before executing it.

The compile phase reads in the JCL file line by line, checking for directly executable lines, keyboard responses, and execution macros. The compile phase also evaluates any compilation statements and writes *all* resultant lines to a file called SYSTEM/JCL. After the compile phase completes, control is normally passed to the execution phase, which executes the SYSTEM/JCL file.

As stated earlier, the JCL works by substituting lines in a file for keyboard entries. However, when you compile a JCL file, it can contain more than just a series of executable commands. (After the compile phase is completed, however, the SYSTEM/JCL file *does* contain only executable lines.)

You can include the following statements in a JCL file you intend to compile:

- Directly executable commands (DIR, BASIC, etc.)
- Pre-arranged keyboard responses
- JCL execution macros (listed in Table 1)
- JCL conditional macros (listed in Table 2)
- Labels
- Another JCL file. When a JCL file "calls" another JCL file, TRSDOS transfers control to the called file and doesn't return control to the calling file.

We use several terms when discussing JCL compilation. They are:

1. Token

A token is a string of up to 8 alphanumeric characters. You can use both upper and lower case letters. Note: There is NO difference between upper and lower case letters for any JCL macro, token, or label.

You can use a token (1) as a true/false switch for logical decisions (see the //IF macro), and (2) as a character string value in substitution fields (see the SUBSTITUTION FIELDS section).

2. Logical operator

The simple logical operators are:

AND (represented by the ampersand symbol "&")
OR (represented by the plus symbol "+")
NOT (represented by the minus symbol "-")

3. Label

A JCL label is used to define the start of a JCL procedure, which allows many small JCL procedures to be combined into one large file. The format for a label is:

```
@label
```

The *label* can be up to 8 alphanumeric characters long.

Table 2/ Conditional Macros

Macro Group	Group Description	Macros	Macro Description
Compilation Comment		//Comment	Acts like a visual log of the JCL file because they are displayed during compilation. These comments are not written to SYSTEM/JCL.
Logical Macros	Define conditional "blocks."	//IF //END //ELSE	Defines the start of a conditional block. Defines the end of a conditional block. Defines the alternative to a false //IF.
Higher Order Logical Macros	Provide for higher conditional logic statements.	//SET //RESET //ASSIGN	Gives a token a logical true value. Gives a token a logical false value. Sets a token's value to true and assigns a character string value to the token.
Termination Macro		//QUIT	Aborts JCL compiling if an invalid condition is detected.
Merge Macro		//INCLUDE	Merges together two or more JCL files during compilation.

Conditional Decisions

Using //IF, //END, //ELSE

The logical compilation macros (//IF, //END, and //ELSE) are used to establish logical "blocks" in a JCL file. When a JCL file is being compiled, these blocks are evaluated as either true or false.

The //IF macro followed by a token determines if the block is true or false.

To set a token true, specify it on the DO command line. To set a token false, do NOT specify it on the DO command line.

These JCL macros produce the following results:

- | | |
|---|---|
| 1) If token is true ... | 2) If token is false ... |
| <pre>//IF token Include these lines. //END</pre> | <pre>//IF token Ignore these lines. //END</pre> |
| 3) If token is false, perform the alternative ... | |
| <pre>//IF token Ignore these lines. //ELSE Include these lines. //END</pre> | |

With this type of logical decision capability, you can create a JCL file and then pick a course of action by typing in a "DO filespec" command with different tokens.

Examples

Consider the following JCL file named START/JCL.

```
. START/JCL for Program start-up
SET *FF to FORMS/FLT
FILTER *PR *FF
//IF PR1
FORMS (CHARS=80)
//ELSE
FORMS (CHARS=132)
//END
```

Assume that these are the first lines in a JCL file that begins execution of an applications program.

To make the //IF PR1 test as true, issue the following DO command:

```
DO START (PR1) ENTER
```

The 80 characters per line is selected.

If (PR1) is not specified on the DO command line, then the //IF test is false and the 132 characters per line is selected.

Using //SET and //RESET

JCL provides the //SET and //RESET macros to reduce the number of tokens in the DO command line.

One basic use for //SET is to let one token set the value of another. For example:

```
//IF KI
//SET P1
//END
```

This JCL file specifies that if KI is true, then P1 is set to a true condition also.

Suppose that the token P2 is already SET and you want to give it a new value. Consider this example:

```
//IF KI
//RESET P2
//END
```

This JCL file specifies that if KI is true, then P2 is reset to a false condition.

Consider the JCL file named MENU/JCL:

```
. MENU/JCL, revision 1
SET *FF TO FORMS/FLT
FILTER *PR *FF
//IF P1
//RESET P2
FORMS (CHARS=80)
//ELSE
//SET P2
FORMS (CHARS=132)
//END
```

If you issue either one of the following commands:

```
DO MENU (P1) (ENTER)
DO MENU (P1,P2) (ENTER)
```

the //IF macro tests P1 as true; therefore P2 is reset to false, and the *FF and 80-character mode are applied.

If you issue either one of the following commands:

```
DO MENU (ENTER)
DO MENU (P2) (ENTER)
```

the //IF macro tests false so the //ELSE macro sets P2 to true and the 132-character mode is applied.

As previously mentioned, the //SET macro can be used to reduce the number of tokens that have to be entered on the DO command line. Consider the following SYSOPT/JCL example:

```
. Establish TRSDOS system options
//IF ALL
//SET COMM
//SET PR
//SET SET
//SET SRES
//END
//IF KIALL
//SET COMM
//SET SET
//END
//IF COMM
set *cl to com/dvr
setcom (word=8)
//END
//IF PR
set *ff to forms/flt
filter *pr *ff
forms (chars=80)
//END
//IF SET
setki (rate=7)
//END
//IF SRES
system (sysres=2)
system (sysres=3)
system (sysres=10)
//END
```

This example shows how many different TRSDOS options can be established with a JCL file. The way it is structured, you can choose any or all of the options.

If you did not use //SET, you would have to enter four separate tokens on the DO command line to establish all of the options, as follows:

```
DO SYSOPT/JCL (COMM,PR,SET,SRES) ENTER
```

If you specify "ALL" in the DO command line, COMM, PR, SET, and SRES are set to true conditions.

If you specify "KIALL" in the DO command line, COMM and SET are set to true conditions.

Notice the use of upper and lower case. As stated earlier, either upper or lower case letters can be used in any JCL macro, token, or label. This is also true when the line is a TRSDOS command, as are the lower case lines in this example.

You can improve the readability of a JCL file by using upper case for macros and lower case for executable lines, such as TRSDOS commands, or vice versa.

Using //ASSIGN

JCL provides the //ASSIGN macro to set a token's logical value true, and to assign a character string value to a token.

The syntax for the //ASSIGN macro is:

```
//ASSIGN token=character string
```

character string can consist of up to 32 characters. Any character on the keyboard is allowed except a double-quotation mark (").

Error Conditions

Any time you use //ASSIGN, there must be at least one character assigned as a value or the compiling aborts.

Examples

In any of the previous examples that used the //SET macro, the //ASSIGN macro could have been substituted. The character string value assigned to the token has no effect on the JCL logic.

In the following example, if the token A is true, the tokens P1, KI, and PR are all set to true. This example assigns character string values to the tokens.

```
.TEST/JCL
//IF A
//ASSIGN P1=PROGRAM/BAS
//ASSIGN KI=ALL
//ASSIGN PR=80
//END
```

Using //. Comment and //QUIT

Compilation comments (//. Comment) are not written to the SYSTEM/JCL file. They are displayed on the screen as they are encountered during compilation. Thus, they act as a visual status log of the compile.

The //QUIT macro aborts the compilation stage if the JCL detects an invalid condition. This macro lets you make sure all needed tokens are entered before any execution takes place.

Examples

```
. START/JCL
set *ff to forms/flt
filter *Pr *ff
forms (lines=60)
//IF KI
setki (rate=5)
//ELSE
//. RATE was not set!
//QUIT
//END
//EXIT
```

If this JCL file is compiled without the token KI being entered on the DO command line, the screen display shows:

```
//. RATE was not set!
//QUIT
```

No actual lines are executed from the SYSTEM/JCL file, because the compile phase was aborted before completion. The compilation comment tells the operator why the abort took place.

If you substitute //ABORT for //QUIT in the previous example and then compile the JCL file without the token KI, the following lines result:

```
//. RATE was not set!
. START/JCL
set *ff to forms/flt
filter *Pr *ff
forms (lines=60)
Job aborted
```

The comment line is displayed as the file is being compiled. However, since //ABORT is an execution macro, the SYSTEM/JCL file finishes compiling and then executes until it reaches the //ABORT line! The //QUIT macro should be used in such a case rather than the //ABORT.

Substitution Fields

One of the most powerful features of the JCL is its ability to substitute and concatenate (add together) character strings to create executable lines.

A substitution field is created by placing pound signs (#) around a token. When the file is compiled, this substitution token is replaced with its current value, either assigned on the DO command line or with the //ASSIGN macro.

Examples

```
. TEST/JCL
set *ff to forms/flt
filter *Pr *ff
forms (chars=#C#)
basic
run"#P1#"
//STOP
```

This example uses two substitution fields: one in the FORMS command line representing the number of characters, and one in the RUN command line.

If you issue the DO command:

```
DO TEST (C=132,P1=PROGRAM1) ENTER
```

the lines written to the SYSTEM/JCL file are:

```
. TEST/JCL
set *ff to forms/flt
filter *Pr *ff
forms (chars=132)
basic
run"PROGRAM1"
//STOP
```

The compile phase substitutes the character string value of the tokens into the actual command line!

The length of the replacement string does not have to be equal to the length of the token name between the # signs.

To reduce the number of tokens needed on the DO command line, and to increase the program options at the same time, use the //ASSIGN macro as follows:

```
. TEST/JCL
//ASSIGN c=80
//ASSIGN P1=Program1
//IF num2
//ASSIGN c=132
//ASSIGN P1=Program2
//END
set *ff to forms/flt
filter *Pr *ff
forms (chars=#C#)
basic
run"#P1#"
//STOP
```

Specifying NUM2 overrides the 80-character printer filter and PROGRAM1 defaults. The values of C and P1 are automatically set with the //ASSIGN tokens inside the //IF conditional block.

Another use for substitution fields is replacing drive numbers.

The following example shows how a FORMAT and BACKUP JCL file can be structured:

```
. FB/JCL, FORMAT with BACKUP
//PAUSE Insert disk to format in drive #D#
format :#D# (name="data1",q=n,ABS)
backup :#S# :#D#
//EXIT
```

The token D represents the destination drive, and the token S represents the source drive.

If you enter the command:

```
DO FB/JCL (S=1,D=2) ENTER
```

the system pauses and prompts you to insert a disk in Drive 2.

Press **ENTER** and the JCL file continues. It formats the disk in Drive 2, and then it executes the backup command with Drive 1 as the source drive and Drive 2 as the destination drive.

The substitution fields can be used in message lines and comments as well as in executable command lines.

Be careful when you want to display a single “#” in a comment or message. Consider the following example:

```
//PAUSE Insert a disk in drive #1
```

If the JCL file were executed only, this line would be properly displayed. However, if the JCL were compiled, an error would occur. For this line to be properly displayed in a compiled JCL, it would have to be written as:

```
//PAUSE Insert a disk in drive ##1
```

Another practical use for substitution fields is copying password protected files from one drive to another.

```
. MOVE/JCL file transfer
copy Program1.#P#:#0 :#D#
copy Program2.#P#:#0 :#D#
copy Program3.#P#:#0 :#D#
copy Program4.#P#:#0 :#D#
//EXIT
```

In this example, a group of files is copied from Drive 0 to a drive specified in the DO command. Also, you have to supply the proper

password for the copies to work. If you specify the wrong password, an error is displayed and the JCL aborts.

Substitution fields can also be concatenated, or added together, to create new fields. For example:

```
. ADD/JCL
COPY #F#/#E# :1
COPY #F1#/#E# :1
//EXIT
```

This example uses two substitution fields, one for the filename and one for the extension.

If you issue the DO command:

```
DO ADD (F=SORT,E=CMD,F1=SORT1) ENTER
```

the following SYSTEM/JCL file results after compiling:

```
. ADD/JCL
COPY SORT/CMD :1
COPY SORT1/CMD :1
//EXIT
```

As in previous examples, the //IF and //ASSIGN macros could be used to allow a single token to select the F, F1, and E tokens.

Combining Files

Most of the JCL examples in the previous sections have been very short. In a practical operating environment, this is often the case. However, each of these small files is taking up the minimum disk allocation of one gran and using one directory entry.

To combine small files and save disk space, use the Label feature of JCL. You can also use the //INCLUDE macro to duplicate a JCL file inside of another JCL file, without having to retype the lines.

Using //INCLUDE

The //INCLUDE macro is used to merge together two or more JCL files during the compile phase. The syntax is:

```
//INCLUDE filespec
```

filespec is a JCL file.

This command is similar to specifying the filespec in a DO command line. However, you cannot enter tokens or other information after the filespec.

If you need to pass tokens to the included program, they will have to be established in the program that is doing the //INCLUDE.

Error Conditions

An `//INCLUDE` macro CANNOT be the last line in a JCL file. If it is, an "End of File Encountered" error occurs, and the JCL aborts.

Examples

This example shows two JCL files and the results of the compile phase. The two JCL files are:

```
. TEST1/JCL          . TEST2/JCL
. comment line 1    . This comment is included
//INCLUDE TEST2
. comment line 2
//EXIT
```

If you issue the command

```
DO TEST1 ENTER
```

the following SYSTEM/JCL file is produced:

```
. TEST1/JCL
. comment line 1
. TEST2/JCL
. This comment is included
. comment line 2
//EXIT
```

The compiling starts with the file named in the DO command line. As soon as the `//INCLUDE` is reached, all lines in the second JCL file are processed, and then the compiling returns to the rest of the original file.

There is no limit to the number of non-nested `//INCLUDE` macros you can use, other than having enough disk space for the resulting SYSTEM/JCL file.

Using JCL Labels

The LABEL feature of JCL allows you to permanently merge together many small JCL procedures into one large file, and then access those procedures individually. This saves disk space and directory entry slots.

Examples

```
. TEST/JCL label example
@FIRST
. this is the first procedure
//exit
@SECOND
. this is the next procedure
@THIRD
. this is the last procedure
```

This file contains three labels. To select any procedure, specify the label on the DO command line.

The following rules determine how much of a labeled JCL file is included in the compile phase:

- 1) If no label is specified on the DO command line, all lines from the beginning of the file up to the first label are compiled.
- 2) If a label is specified, compiling includes all lines from the specified label until the next label or the end of the file is reached.

DOing the TEST/JCL file using the @FIRST label would write the comment “. this is the first procedure” and the //EXIT macro to the SYSTEM/JCL file for execution. Specifying either of the other labels would include only the appropriate single comment line.

If you compiled the file without specifying a label in the DO command, only the initial execution comment “. TEST/JCL label example” would be written in SYSTEM/JCL.

There is no limit to the size of a labeled procedure. They can range from one to as many lines as you can fit on your disk. The only requirement is that a JCL file containing labels must be compiled.

When you use labels in a JCL file, we recommend that you start the file with a comment line or some executable line other than a label.

Suppose @FIRST is the first line in the following file:

```
@FIRST
. Print this comment
```

If you issued a DO command for this file without specifying the @FIRST label, the compiling phase would receive the first line, see that it is a label, and quit. Since the compile is complete, the SYSTEM/JCL file would be executed! And since nothing was written to SYSTEM/JCL, its old contents are not erased. In other words, whatever lines had been compiled to the SYSTEM/JCL file from a previous DO command would now be executed.

Advanced JCL Compiling

The previous section on JCL compiling described the basic uses of tokens and compilation macros. If you do not understand the JCL Compiling section, please re-read it. If you actually type in and try the examples, you will get a better understanding of how to structure a JCL file for compiling.

This section describes additional features and shows different ways to accomplish logical decision branching. These additional features are explained in four parts:

-
- Using the Logical Operators
 - Using Nested //IF Macros
 - Using Nested //INCLUDE Macros
 - Using the Special % Symbol

Using the Logical Operators

The logical operators used with the //IF macro (AND, OR, and NOT) specify the type of logical testing, and they are represented as follows:

AND — ampersand (&)
OR — plus sign (+)
NOT — minus sign (-)

All previous examples of //IF tested the logical truth or falseness of a token. You can accomplish more complex and efficient testing by using the logical operators.

Consider the following series of examples using the tokens A and B:

```
//IF -A
. include these lines if A is not specified
//END
```

By using NOT (-), you can see if a token is false, which provides an alternative method to select a block of lines for compiling.

```
//IF A+B
. include these lines if A or B is specified
//END
```

```
//IF A&B
. include these lines if A and B are specified
//END
```

These examples show how multiple tokens may be tested in a single //IF statement. The first example is true if either A OR B is true. The second example is true only if both A AND B are true.

You can use any combination of logical operators in an //IF statement. The following rules apply:

- The expressions are evaluated from left to right.
- Do not use parentheses because they abort the JCL compiling.
- All logical operators have the same priority.

You can combine the logical operators to test almost any arrangement of tokens. You can combine the logical operators to set up default conditions and to check for missing tokens, as the following examples demonstrate.

```

. CHECK/JCL          . CHECK1/JCL
//IF -S              //IF -S+-D
//ASSIGN S=0        //, You MUST enter S and D!
//END                //QUIT
//IF -D              //END
//ASSIGN D=2
//END

```

The CHECK example tests S and D individually, and assigns them default values if they were not true (that is, if they were not specified in the DO command line).

The CHECK1 example is structured so that both S and D must be true (specified on the DO command line), or the JCL compiling aborts.

Using Nested //IF Macros

By definition, a conditional block begins with an //IF and concludes with an //END.

When the //IF evaluates true, the lines between the //IF and the //END or an //ELSE (if one exists) are compiled. It is also possible to include other //IF - //END blocks within the main conditional block (called nesting).

The //ELSE macro provides an alternative course of action in case an //IF evaluates false. It is also possible to have more //IF - //END statements following the //ELSE. Refer to the following examples:

```

. TEST/JCL
//IF A
. comment 1
//ELSE
//IF B
. comment 2
//END      (ends the //IF B statement)
//END      (ends the //IF A statement)

```

If A evaluates true, comment 1 is written out, and the //ELSE is ignored. If A is false, B is tested. The comment 2 is written out only if B is true. Notice the two //END macros. There must be one //END for every //IF.

You can document your own JCL files in the same way that we have documented these examples.

Documenting //END macros increases the readability of the files, especially when you edit a file that you have created some weeks (or months) previously.

```

//IF A
. Comment A
//IF B
. Comment B
//IF C
. Comment C
//END          (ends Third IF)
//END          (ends Second IF)
. Comment D
//END          (ends First IF)

```

If the first //IF is false, all lines up to the corresponding //END are ignored.

If the first //IF is true, Comment A and Comment D are written to SYSTEM/JCL.

If //IF B is true, Comment B is also written to SYSTEM/JCL. If B is false, all lines up to the corresponding //END are ignored.

The only time //IF C is considered is if both A and B test true. If C is true, Comments A through D are written to SYSTEM/JCL.

Although not shown in the example, you can use the logical operators when nesting //IFs.

Using Nested //INCLUDE Macros

When you use the //INCLUDE macro, the included file can also contain another //INCLUDE macro. This is called nesting. The following rules apply:

- The maximum nest level is five active //INCLUDE macros.
- An //INCLUDE macro cannot be the last line in a JCL file.

Example

The following example uses three files to show how the lines in nested //INCLUDE files are processed:

```

//. NEST0/JCL
. nested procedure example (Nest 0)
//INCLUDE nest1
. this is the end of the primary JCL (Nest 0)
//EXIT

//. NEST1/JCL
. this is the first nest (Nest 1)
//INCLUDE nest2
. this is the end of the first nest (Nest 1)

//. NEST2/JCL
. this is the second nest (Nest 2)

```

If you save these JCL files as NEST0/JCL, NEST1/JCL, and NEST2/JCL and then compile and execute NEST0/JCL, the following SYSTEM/JCL results:

```
//. NEST0/JCL
//. NEST1/JCL
//. NEST2/JCL
. nested procedure example (Nest 0)
. this is the first nest (Nest 1)
. this is the second nest (Nest 2)
. this is the end of the first nest (Nest 1)
. this is the end of the primary JCL (Nest 0)
```

The //INCLUDE macro can be used to compile a large JCL procedure from a series of smaller JCL routines. If the finished SYSTEM/JCL file is a procedure that will be executed many times, you can easily save it by copying SYSTEM/JCL to a file with another name.

Using the Special % Symbol

The % symbol is used to pass character values (in hex) to the system as though they came from the keyboard. The syntax is:

%character value

Below are some valid values and their results:

Hex Value	Result
09	Position to next tab stop (every 8 columns)
0A	Linefeed
1F	Clear screen

The value of any printable character can also be used, although control characters (characters with a value less than hex 20) are generally used. (See Appendix C for a list of characters, values, and actions performed on the video display.)

Examples

You should place the clear screen character at the start of a line. For example:

```
%1F//PAUSE Insert disk in drive 1, press ENTER
```

clears the screen and displays the JCL line in the top left corner of the screen.

The tab and linefeed characters, used to position comments or lines on the screen, should always be placed AFTER the period in the comment line or the macro in an executable line. For example:

```
.%09%09 This comment is positioned at the
second tab stop.
//PAUSE %0A%0A%0A This line appears 3 lines down
```

If you place the character BEFORE the period, TRSDOS does not recognize it as a comment line and the JCL aborts.

If you place the character AFTER the macro, the //PAUSE is displayed and the remaining message line is displayed 3 lines lower on the screen.

Using the tab and linefeed characters in this manner can sometimes help to improve the readability of the messages displayed during JCL execution.

Using TRSDOS JCL To Interface With Applications Programs

This appendix describes how to use JCL to start up and control your applications programs.

Two languages are discussed: BASIC and Z-80 assembly.

Interfacing With BASIC

A JCL file is the perfect method to interface between the operating system and the BASIC language. JCL can be used to create procedures that require only the inserting of a diskette to start up a program. Additionally, you can utilize the features of JCL from within a BASIC program.

Examples

To use a JCL file to initiate an automatic start-up of a BASIC program, you can use the AUTO library command to execute a JCL file.

Assuming the JCL file is named BAS/JCL, issuing the command:

```
AUTO DD BAS/JCL (ENTER)
```

automatically executes the desired BASIC program every time the computer is booted with the AUTOed system disk.

In order to execute a BASIC program from a JCL file, lay out the JCL file as follows:

1. Establish any necessary drivers, filters, or other TRSDOS options.
2. Enter BASIC with any necessary parameters (such as memory size and number of files).
3. RUN the BASIC program.

-
4. Terminate the JCL execution with //STOP (which leaves control with BASIC).

You can also enter a DO command directly from the TRSDOS Ready prompt to execute a BASIC program.

To execute a JCL file once you have entered BASIC, the command format is:

```
SYSTEM"DO filename"
```

This command can be typed in directly or entered as a BASIC program line.

Also, any JCL file called from BASIC should contain the //EXIT termination macro, so that control will return to TRSDOS Ready when the JCL file is completed.

For example, suppose you want to use the JCL //ALERT macro to inform you when a lengthy BASIC procedure has completed. Following the lines containing the BASIC procedure, you could have a BASIC program line such as:

```
1000 SYSTEM "DO = ALERT/JCL:0"
```

which executes the ALERT/JCL file:

```
. Your procedure is complete. Press ENTER to  
resume.  
//ALERT (1,0,7,0)  
BASIC  
//STOP
```

When BASIC reaches line 1000, the JCL file ALERT/JCL is executed, sending a series of repeating tones out the tone generator.

You are notified that your BASIC procedure has completed. Pressing **ENTER** ends the JCL alert and returns you to BASIC.

There are two important points about this example. First, the comment line in the ALERT/JCL file is absolutely necessary, as a JCL file cannot start with an execution macro. Second, the "BASIC" statement will reload BASIC. If you want a particular program to be loaded and run, you can place its name on the command line or add the BASIC commands before the //STOP statement. The //STOP termination macro must be included to assure that keyboard control remains within BASIC.

Although the example demonstrates an execute only JCL file, you can also call compiled JCL procedures from BASIC. You can even construct a SYSTEM "DO filespec [(parameters)]" command using BASIC string substitution.

Any time you want to use a SYSTEM "DO filespec" command from BASIC to execute another BASIC program, you have to change the format of the command. To DO these types of JCL files from BASIC, use the commands:

```
SYSTEM ENTER  
DO filespec [(parameters)] ENTER
```

Using this format for the command assures that a proper exit is made before the new JCL file is started.

Controlling a BASIC program

In some cases, the prompts in a BASIC program can be answered with a line from a JCL file. This is true if the program uses the INPUT or LINEINPUT BASIC statement to take the input.

If the program uses the INKEY\$ statement, response has to come from the keyboard rather than from a JCL file. If the program uses the proper input method, you can create a JCL for total hands-off operation as follows:

1. Run through the BASIC program, making a note of every prompt to be answered.
2. Create a JCL file to enter BASIC and run the program as explained above in the BAS/JCL example. Leave off the //STOP macro.
3. Add the responses to the prompts as lines in the JCL file.

Using this method provides automatic program execution. Terminating the JCL file depends on what needs to be done when the application program has completed.

If you want to run more programs, you could add the proper RUN"PROGRAM" line to the JCL file, followed by any required responses to program prompts.

If you want to return to the TRSDOS Ready mode, you could end the file with the //EXIT macro. If you want to return to the BASIC Ready mode, you could end the file with the //STOP macro.

Interfacing With Z-80 ASSEMBLY

It is very simple to interface an assembly language program with the DO processor. All programs that utilize the line input handler (identified as the @KEYIN supervisor call the the "Technical Information" manual) are able to accept "keyboard" input from the JCL file, just as though you typed it in when the program ran.

This gives the capability of pre-arranging the responses to a program's requests for input, inserting the responses into the JCL file, initiating the procedure, then walking away from the machine while it goes about its business of running the entire job.

Keyboard input normally handled by the single-entry keyboard routines (@KBD, @KEY, and BASIC's INKEY\$) continue to be requested from the keyboard at program run time and do not utilize the JCL file data for input requests.

Practical Examples Of TRSDOS JCL Files

It is virtually impossible to show all the many uses of JCL files.

We give you two examples of how you can make your day-to-day TRSDOS operations even more efficient using JCL files.

1)

This example shows how to SYSRES system modules using a JCL file. The modules to be resided are 2, 3, and 10. These modules have to be resident in memory to perform a backup by class between two non-system diskettes in a two-drive system.

The JCL file to SYSRES these modules may look something like this:

```
. BURES/JCL - JCL used to SYSRES modules 2, 3,
and 10
SYSTEM (SYSRES=2
SYSTEM (SYSRES=3
SYSTEM (SYSRES=10
. end of BURES/JCL
```

When executed, this JCL file causes the system modules 2, 3, and 10 to be resided in high memory. Because this JCL uses no labels or compilation macros, the compilation phase can be skipped.

2)

This example shows how to back up a diskette using a JCL file.

A minimum of three drives are required. Drive 0 must contain a system diskette with the JCL file. Drive 1 contains the source diskette. Assume that the source diskette's name is MYDISK and its master password is PASSWORD. Also, assume that it is 40 track, and double density. Drive 2 contains the destination diskette.

The JCL file to perform the backup may look something like this:

```
. DUPDISK/JCL - Disk duplication JCL
//PAUSE Source in 1, Dest. in 2, ENTER when
ready
format :2 (name="mydisk",q=n,abs)
//PAUSE format ok? ENTER if yes, BREAK if no
backup :1 :2
. end of backup - will now restart JCL
do *
```

The second line of the JCL causes the computer to pause until the **ENTER** key is pressed. This allows you to insert the proper diskette into Drives 1 and 2. Once you insert the proper diskettes, press **ENTER** and the third line of the JCL is executed.

The format line passes the NAME parameter to the format utility. Note that the diskette name, and diskette password of the destination

diskette must be an exact match of the source disk. If they do not exactly match, the JCL aborts.

Also, note that the parameters Q=N and ABS are specified. Both are necessary. The Q=N parameter causes the computer to use the default of PASSWORD for the master password, by passing the "Master Password" prompt. The ABS parameter ensures that no prompt appears if the destination diskette contains data.

The pause after the format statement allows you to check whether or not the format is successful. If the destination diskette is properly formatted, press **(ENTER)** to continue the JCL.

After you press **(ENTER)** in response to the seconds pause, the backup takes place. When the backup completes, the comment line appears, and the DO * command executes. The command causes the SYSTEM/JCL file to execute. Realize that since this is a repeating JCL, the compilation phase cannot be skipped.

If tracks are locked out during the format, press **(BREAK)**. Pressing **(BREAK)** aborts the JCL, and you have to restart the JCL file.

Important: Be aware that if BACKUP or FORMAT is being executed by a JCL file, the following rules apply:

1. If the backup is mirror image, the source and destination disk Disk ID's must be the same or the backup aborts.
2. Backups with the (X) parameter, single-drive backups, and backups with the (QUERY) parameter are not allowed.
3. Single-drive formats are not allowed.

Appendix B/ Model 4/4P Hardware

The Keyboard Code Map

The keyboard code map shows the code that TRSDOS returns for each key, in each of the modes: control, shift, unshift, clear and control, clear and shift, clear and unshift.

For example, pressing **CLEAR**, **SHIFT**, and **1** at the same time returns the code X'A1'.

A program executing under TRSDOS — for example, BASIC — may translate some of these codes into other values. Consult the program's documentation for details.

BREAK Key Handling

The **BREAK** key (X'80') is handled in different ways, depending on the settings of three system functions. The table below shows what happens for each combination of settings.

Break Enabled	Break Vector Set	Type-Ahead Enabled	
Y	N	Y	If characters are in the type-ahead buffer, then the buffer is emptied. * If the type-ahead buffer is empty, then a BREAK character (X'80') is placed in the buffer. *
Y	N	N	A BREAK character (X'80') is placed in the buffer.
Y	Y	Y	The type-ahead buffer is emptied of its contents (if any), and control is transferred to the address in the BREAK vector (see @BREAK SVC). *
Y	Y	N	Control is transferred to the address in the BREAK vector (see @BREAK SVC).
N	X	X	No action is taken and characters in the type-ahead buffer are not affected.

Y means that the function is on or enabled
N means that the function is off or disabled
X means that the state of the function has no effect

Break is enabled with the SYSTEM (BREAK = ON) command (this is the default condition).

The break vector is set using the @BREAK SVC (normally off).

Type-ahead is enabled using the SYSTEM (TYPE = ON) command (this is the default condition).

* Because the **BREAK** key is checked for more frequently than other keys on the keyboard, it is possible for **BREAK** to be pressed after another key on the keyboard and yet be detected first.

Specifications

Model 4

The Radio Shack TRS-80 Model 4 is a ROM/disk-based computer system with one major part:

- A display console/keyboard unit with one or two built-in, single-sided, double-density, floppy disk drives (disk system) or zero disk drives (cassette system)

The operating system software is loaded from ROM or an operating system disk in Drive 0 by a built-in read-only memory (ROM) "bootstrap" program.

Model 4P

The Radio Shack TRS-80 Model 4P is a disk-based computer system with one major part:

- A display console/keyboard unit with two built-in, single-sided, double-density, floppy disk drives

The operating system software is loaded from an operating system disk in Drive 0 by a built-in read-only memory (ROM) "bootstrap" program.

Console

Processor

Model 4

- The TRS-80 Model 4 is a Z-80A based high-speed microprocessor with 64K or optional 128K of memory (disk system) or 16K of memory (cassette system)
- The processor receives power-up and reset instructions from ROM.
- The Model 4 is compatible with existing Model III software.

Model 4P

- The TRS-80 Model 4P is a Z-80A based high-speed microprocessor with 64K or optional 128K of memory
- The processor receives power-up and reset instructions from ROM
- The Model 4P is compatible with existing Model III disk software.

Sound

The disk system can generate software-controlled tones, one at a time.

Video Display

Six Modes

- White on black (normal)
- Black on white (reversed)
- 64 characters by 16 lines Model III Mode
- 32 characters by 16 lines Model III Mode
- 80 characters by 24 lines Model 4 Mode
- 40 characters by 24 lines Model 4 Mode

Displayable Characters

- Full ASCII set
- 64 graphics characters

Keyboard

The keyboard has the standard typewriter keys, numeric keypad, and three function keys.

Three Modes

- Control
- Shift
- Caps

Floppy Disk Drives

Minimum

Model 4 : One built-in 5-1/4-inch, single-sided floppy drive (disk system) or zero drives (cassette system)

Model 4P: Two built-in 5-1/4-inch, single-sided floppy drives

Maximum

Model 4 : Two built-in and two external 5-1/4-inch, single-sided floppy drives

Model 4P: Two built-in 5-1/4-inch, single-sided floppy drives

Preventive Maintenance Interval

- Typical usage (3,000 power-on hours per year): Every 8,000 power-on hours
- Heavy usage (8,000 power-on hours per year): Every 5,000 power-on hours

Required Media

- Radio Shack single-sided, 5-1/4-inch floppy disks

The Data Transfer Rate is 250K bits per second.

Power Supply

Power Requirements

- 105-130 VAC, 60 Hz
- 240 VAC, 50 Hz (Australian)
- 220 VAC, 50 Hz (European)
- Grounded outlet

Maximum Current Drain

- 1.7 Amps

Typical Current Drain

- 1.5 Amps

Operating Temperature

- 55 to 80 degrees Fahrenheit
- 13 to 27 degrees Centigrade

Peripheral Interfaces

Standard

- Floppy disk input/output channel for connection of one or two external floppy disk drives (Model 4 only)
- I/O bus for connection of hard disk and other peripherals
- Cassette I/O jack (Model 4 only)

Optional

- High-resolution graphics board
- Serial port RS-232C
- Auto answer modem (Model 4P only)

Serial Interface

One Port:

- Allows asynchronous or synchronous transmission
- Conforms to the RS-232C standard
- Uses the DB-25 connector on the bottom of the Model 4 display console and on the back of the Model 4P

The DB-25 connector pin-outs and signals available are listed below:

Signal	Function	Pin#
PGND	Protective Ground	1
TD	Transmit Data	2
RD	Receive Data	3
RTS	Request to Send	4
CTS	Clear to Send	5
DSR	Data Set Ready	6
SGND	Signal Ground	7
CD	Carrier Detect	8
DTR	Data Terminal Ready	20
RI	Ring Indicator	22
STD†	Secondary Transmit Data (Model 4 only)	14
SUN†	Secondary Unassigned (Model 4 only)	18
SRTS†	Secondary Request to Send	19

† These signals are not used for secondary functions but are reserved for future use.

Parallel Interface

- Connection to a line printer via the 34-pin connector on the bottom of the Model 4 display console and on the back of the Model 4P.
- Eight data bits are output in parallel
- Eight data bits are input
- All levels are TTL compatible

The parallel printer pin-outs and signals available are listed below.

NOTE: If a signal name contains an asterisk (*), the signal is active-low.

Signal	Function	Pin #
STROBE*	1.5 microseconds pulse to clock the data from processor to printer	1
DATA 0	Bit 0 (lsb) of output data byte	3
DATA 1	Bit 1 of output data byte	5
DATA 2	Bit 2 of output data byte	7
DATA 3	Bit 3 of output data byte	9
DATA 4	Bit 4 of output data byte	11
DATA 5	Bit 5 of output data byte	13
DATA 6	Bit 6 of output data byte	15
DATA 7	Bit 7 (msb) of output data byte	17
BUSY	Input to computer from printer, high indicates busy	21
PAPER EMPTY	Input to computer from printer, high indicates no paper — If the printer doesn't provide this, the signal is forced low	23
BUSY* †	Inverse of BUSY (Pin 2)	25
FAULT*	Input to computer from printer, low indicates fault (paper empty, ribbon out, printer off-line, and so on)	28
GROUND	Common signal ground	2,4,6,8,10 12,14,16, 18,20,22, 24,27,31, 33
NC	Not connected or not used	19,26,29, 30,32,34

† Depending on the kind of printer used, this signal may be called "UNIT SELECT." See your printer manual for more information.

Communications

For hardwiring two Model 4/4P's without a modem, use Radio Shack's RS-232C cables (cat. no. 26-1490, -1491, -1492, -1493) and null modem adapter (cat. no. 26-1496).

Appendix C/ Character Codes

Text, control functions, and graphics are represented in the computer by codes. The character codes range from zero through 255.

Code 0 is a prefix code. It tells the video driver to display the special character for codes 1 - 31. These codes are normally treated as cursor control commands.

Codes 1 through 31 normally represent certain control functions. For example, code 13 represents a carriage return or "end of line." These same codes also represent special display characters. To display the special character that corresponds to a particular code (1 - 31), precede the code with a code zero. (Note: Some screen control characters cannot be entered from the TRSDOS Ready or BASIC Ready prompts, but can be directed to the screen by program control. See the CHR\$ and ASC functions in the BASIC portion of this manual.)

Codes 32 through 127 represent the text characters — all those letters, numbers, and other characters that are commonly used to represent textual information.

Codes 128 through 191, when output to the video display, represent 64 graphics characters.

Codes 192 through 255, when output to the video display, represent either space compression codes or special or alternate characters, as determined by software. Toggling between these modes is done via codes 21 and 22.

Code 21 toggles the video driver between space compression codes and the special/alternate character set. Code 22 toggles the video driver between the special character set and the alternate character set. The setting of the toggle controlled by code 21 determines if the code 22 toggle will have any effect on what is subsequently displayed.

The following chart illustrates the power-up and first toggle states for codes 21 and 22:

	Code 21	Code 22
Power-up state	space compression characters	special characters
First toggle state	special/alternate characters	alternate characters

At power-up, codes in the range 192 to 255 will produce one or more spaces (space compression mode). From this point, you can enter the special character set by outputting a code 21 to the display. You can then enter the alternate character set by outputting a code 22 to the display. To switch back to the special set, output another code 22. To switch back to space compression codes from either the special or alternate character set, output a code 21.

When you are in space compression mode, outputting a code 22 still toggles between special and alternate character sets, even though it does not affect the characters subsequently displayed. Any characters in the range 192-255 that are already on the display will toggle between special and alternate character sets each time a code 22 is received.

Note: Special and alternate characters are not available if reverse video (code 16) is enabled.

ASCII Character Set

Code		ASCII		Video Display
Dec.	Hex.	Abbrev.	Keyboard	
0	00	NUL	CTRL @	Next character is treated as displayable; if in the range 1 - 31, a special character is displayed (see list of special characters later in this Appendix)
1	01	SOH	CTRL A	
2	02	STX	CTRL B	
3	03	ETX	CTRL C	
4	04	EOT	CTRL D	
5	05	ENQ	CTRL E	
6	06	ACK	CTRL F	
7	07	BEL	CTRL G	
8	08	BS	Left Arrow CTRL H	Backspace and erase
9	09	HT	Right Arrow CTRL I	Move cursor to the next tab stop (located every 8 columns)
10	0A	LF	Down Arrow CTRL J	Move cursor to start of next line
11	0B	VT	Up Arrow CTRL K	
12	0C	FF	CTRL L	
13	0D	CR	ENTER CTRL M	Move cursor to start of next line
14	0E	SO	CTRL N	Cursor on
15	0F	SI	CTRL O	Cursor off

Code		ASCII			Video Display
Dec.	Hex.	Abbrev.	Keyboard		
16	10	DLE	CTRL P		Enable reverse video and set high bit routine on*
17	11	DC1	CTRL Q		Set high bit routine off*
18	12	DC2	CTRL R		
19	13	DC3	CTRL S		
20	14	DC4	CTRL T		
21	15	NAK	CTRL U		Swap space compression/ special characters
22	16	SYN	CTRL V		Swap special/alternate characters
23	17	ETB	CTRL W		Set to 40 characters per line
24	18	CAN	SHIFT ← CTRL X		Backspace without erasing
25	19	EM	SHIFT → CTRL Y		Advance cursor
26	1A	SUB	SHIFT ↓ CTRL Z		Move cursor down
27	1B	ESC	SHIFT ↑ CTRL ,		Move cursor up
28	1C	FS	CTRL /		Move cursor to upper left corner. Disable reverse video and set high bit routine off.* Set to 80 characters per line.
29	1D	GS	CTRL ENTER		Erase line and start over
30	1E	RS	CTRL ;		Erase to end of line
31	1F	VS	SHIFT CLEAR		Erase to end of display
32	20	SPA	SPACE BAR		(blank)
33	21		!		!
34	22		"		"
35	23		#		#
36	24		\$		\$

* When the high bit routine is on, characters 20 - 127 are converted to characters 148 - 255. When reverse video is enabled, characters 128 - 191 are displayed as standard ASCII characters in reverse video.

Code		Keyboard	Video Display
Dec.	Hex.		
37	25	%	%
38	26	&	&
39	27	,	,
40	28	((
41	29))
42	2A	*	*
43	2B	+	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	:	:
59	3B	;	;
60	3C	<	<
61	3D	=	=
62	3E	>	>
63	3F	?	?
64	40	@	@
65	41	*A	A

* A - Z (codes 65 - 90) are shifted functions. Hold down **SHIFT** and then press the desired key.

Code		Keyboard	Video Display
Dec.	Hex.		
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	CLEAR ,	[
92	5C	CLEAR /	\
93	5D	CLEAR .]
94	5E	CLEAR ;	^
95	5F	CLEAR ENTER	_
96	60	SHIFT @	'

Code		ASCII	Keyboard	Video Display
Dec.	Hex.	Abbrev.		
97	61		A	a
98	62		B	b
99	63		C	c
00	64		D	d
101	65		E	e
102	66		F	f
103	67		G	g
104	68		H	h
105	69		I	i
106	6A		J	j
107	6B		K	k
108	6C		L	l
109	6D		M	m
110	6E		N	n
111	6F		O	o
112	70		P	p
113	71		Q	q
114	72		R	r
115	73		S	s
116	74		T	t
117	75		U	u
118	76		V	v
119	77		W	w
120	78		X	x
121	79		Y	y
122	7A		Z	z
123	7B		CLEAR SHIFT ,	{
124	7C		CLEAR SHIFT /	
125	7D		CLEAR SHIFT .	}
126	7E		CLEAR SHIFT ;	~
127	7F	DEL	CLEAR SHIFT ENTER	±

Extended (non-ASCII) Character Set

Code		ASCII	Video Display
Dec.	Hex.	Keyboard	
128	80	BREAK	See Special Character Table
129	81	CLEAR CTRL A F1	"
130	82	CLEAR CTRL B F2	"
131	83	CLEAR CTRL C F3	"
132	84	CLEAR CTRL D	"
133	85	CLEAR CTRL E	"
134	86	CLEAR CTRL F	"
135	87	CLEAR CTRL G	"
136	88	CLEAR CTRL H	"
137	89	CLEAR CTRL I	"
138	8A	CLEAR CTRL J	"
139	8B	CLEAR CTRL K	"
140	8C	CLEAR CTRL L	"
141	8D	CLEAR CTRL M	"
142	8E	CLEAR CTRL N	"
143	8F	CLEAR CTRL O	"
144	90	CLEAR CTRL P	"
145	91	CLEAR CTRL Q SHIFT F1	"
146	92	CLEAR CTRL R SHIFT F2	"
147	93	CLEAR CTRL S SHIFT F3	"
148	94	CLEAR CTRL T	"
149	95	CLEAR CTRL U	"
150	96	CLEAR CTRL V	"
151	97	CLEAR CTRL W	"
152	98	CLEAR CTRL X	"

Code		ASCII	Video Display
Dec.	Hex.	Keyboard	
153	99	CLEAR CTRL Y	See Special Character Table
154	9A	CLEAR CTRL Z	"
155	9B	CLEAR SHIFT up arrow	"
156	9C		"
157	9D		"
158	9E		"
159	9F		"
160	A0	CLEAR SPACE	"
161	A1	CLEAR SHIFT 1	"
162	A2	CLEAR SHIFT 2	"
163	A3	CLEAR SHIFT 3	"
164	A4	CLEAR SHIFT 4	"
165	A5	CLEAR SHIFT 5	"
166	A6	CLEAR SHIFT 6	"
167	A7	CLEAR SHIFT 7	"
168	A8	CLEAR SHIFT 8	"
169	A9	CLEAR SHIFT 9	"
170	AA	CLEAR SHIFT :	"
171	AB		"
172	AC		"
173	AD	CLEAR -	"
174	AE		"
175	AF		"
176	B0	CLEAR 0	"
177	B1	CLEAR 1	"
178	B2	CLEAR 2	"
179	B3	CLEAR 3	"
180	B4	CLEAR 4	"
181	B5	CLEAR 5	"
182	B6	CLEAR 6	"

Code Dec.	Code		ASCII Keyboard	Video Display
	Hex.	Hex.		
183	B7		CLEAR 7	See Special Character Table
184	B8		CLEAR 8	"
185	B9		CLEAR 9	"
186	BA		CLEAR :	"
187	BB			"
188	BC			"
189	BD		CLEAR SHIFT -	"
190	BE			"
191	BF			"
192	C0		CLEAR @ *	"
193	C1		CLEAR A **	"
194	C2		CLEAR B **	"
195	C3		CLEAR C **	"
196	C4		CLEAR D **	"
197	C5		CLEAR E **	"
198	C6		CLEAR F **	"
199	C7		CLEAR G **	"
200	C8		CLEAR H **	"
201	C9		CLEAR I **	"
202	CA		CLEAR J **	"
203	CB		CLEAR K **	"
204	CC		CLEAR L **	"
205	CD		CLEAR M **	"
206	CE		CLEAR N **	"
207	CF		CLEAR O **	"
208	D0		CLEAR P **	"
209	D1		CLEAR Q **	"
210	D2		CLEAR R **	"

* Empties the type-ahead buffer.

** Used by Keystroke Multiply, if KSM is active.

































































Code Dec.	Code Hex.	ASCII Keyboard	Video Display
211	D3	CLEAR S **	See Special Character Table
212	D4	CLEAR T **	"
213	D5	CLEAR U **	"
214	D6	CLEAR V **	"
215	D7	CLEAR W **	"
216	D8	CLEAR X **	"
217	D9	CLEAR Y **	"
218	DA	CLEAR Z **	"
219	DB		"
220	DC		"
221	DD		"
222	DE		"
223	DF		"
224	E0	CLEAR SHIFT @	"
225	E1	CLEAR SHIFT A	"
226	E2	CLEAR SHIFT B	"
227	E3	CLEAR SHIFT C	"
228	E4	CLEAR SHIFT D	"

** Used by Keystroke Multiply, if KSM is active.

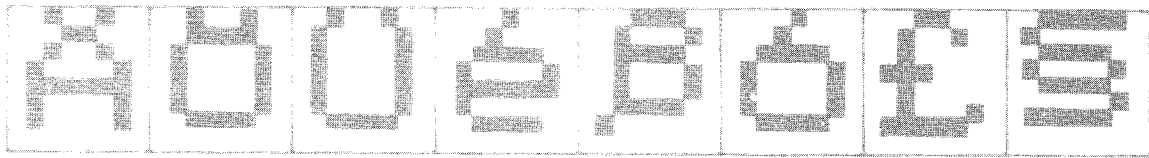
Code Dec.	Code		ASCII Keyboard	Video Display
	Hex.			
229	E5		CLEAR SHIFT E	See Special Character Table
230	E6		CLEAR SHIFT F	"
231	E7		CLEAR SHIFT G	"
232	E8		CLEAR SHIFT H	"
233	E9		CLEAR SHIFT I	"
234	EA		CLEAR SHIFT J	"
235	EB		CLEAR SHIFT K	"
236	EC		CLEAR SHIFT L	"
237	ED		CLEAR SHIFT M	"
238	EE		CLEAR SHIFT N	"
239	EF		CLEAR SHIFT O	"
240	F0		CLEAR SHIFT P	"
241	F1		CLEAR SHIFT Q	"
242	F2		CLEAR SHIFT R	"
243	F3		CLEAR SHIFT S	"
244	F4		CLEAR SHIFT T	"
245	F5		CLEAR SHIFT U	"
246	F6		CLEAR SHIFT V	"
247	F7		CLEAR SHIFT W	"

Code		ASCII Keyboard	Video Display
Dec.	Hex.		
248	F8	CLEAR SHIFT X	See Special Character Table
249	F9	CLEAR SHIFT Y	"
250	FA	CLEAR SHIFT Z	"
251	FB		"
252	FC		"
253	FD		"
254	FE		"
255	FF		"

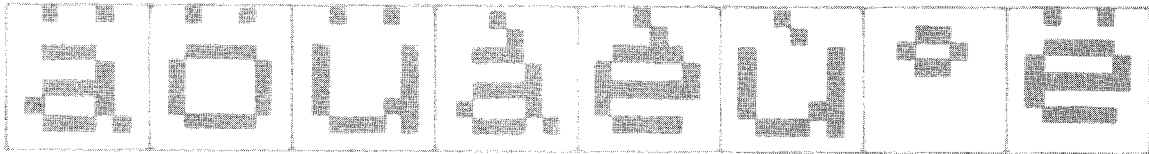
Graphics Characters (Codes 128-191)

	128		136		144		152		160		168		176		184
	129		137		145		153		161		169		177		185
	130		138		146		154		162		170		178		186
	131		139		147		155		163		171		179		187
	132		140		148		156		164		172		180		188
	133		141		149		157		165		173		181		189
	134		142		150		158		166		174		182		190
	135		143		151		159		167		175		183		191

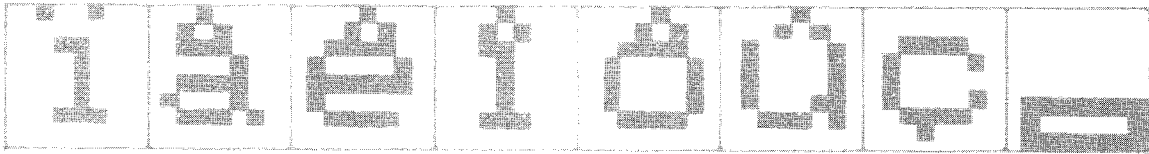
Special Characters (0-31, 192-255)



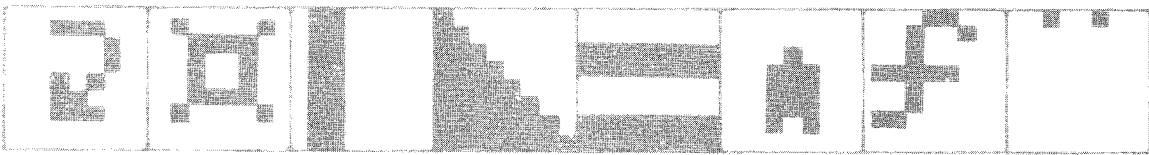
0 1 2 3 4 5 6 7



8 9 10 11 12 13 14 15



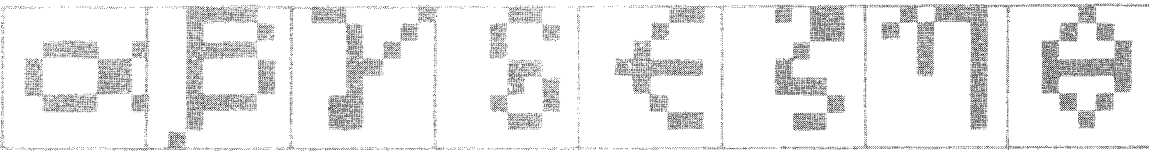
16 17 18 19 20 21 22 23



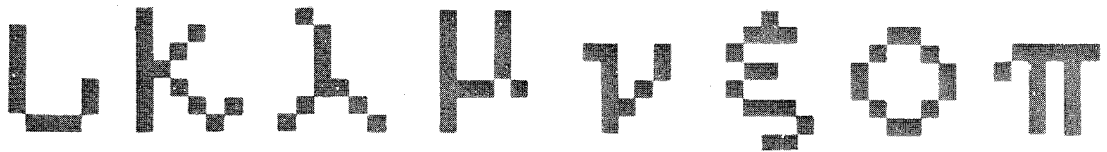
24 25 26 27 28 29 30 31



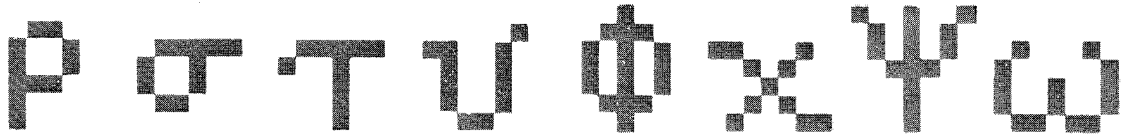
192 193 194 195 196 197 198 199



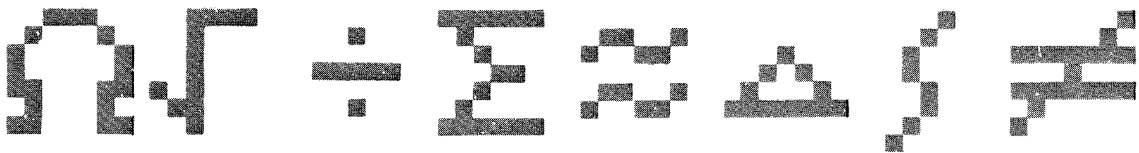
200 201 202 203 204 205 206 207



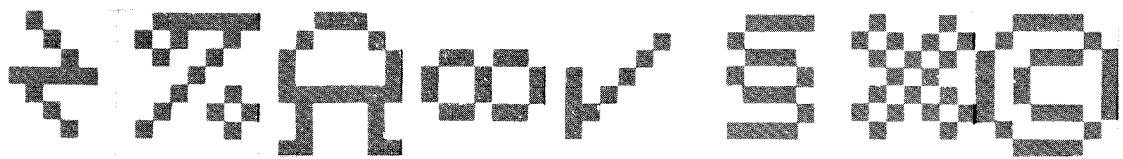
208 209 210 211 212 213 214 215



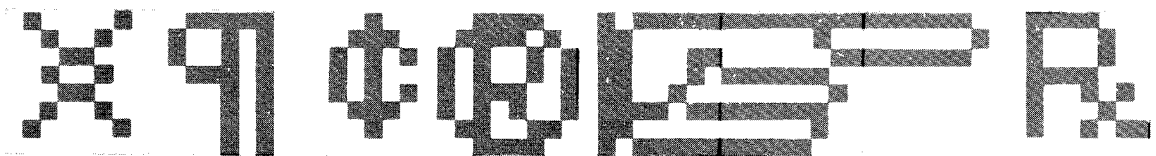
216 217 218 219 220 221 222 223



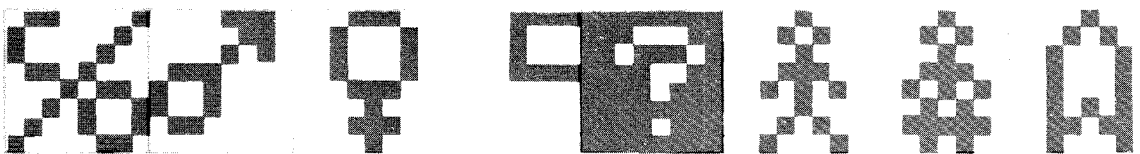
224 225 226 227 228 229 230 231



232 233 234 235 236 237 238 239



240 241 242 243 244 245 246 247



248 249 250 251 252 253 254 255

Appendix D/ Error Messages and Problems

In Case Of Difficulty

Your TRSDOS operating system was designed and tested to provide you with trouble-free operation. If you do experience problems, there is a good chance that something other than the TRSDOS system is at fault. This section discusses some of the most common user problems, and suggests general cures for these problems.

Problem 1 . . . The system seems to access the wrong disk drives, or cannot read the diskettes.

If you have trouble reading Model I and III TRSDOS diskettes, refer to the REPAIR and CONV Utilities. Those sections explain how to make these types of disks readable.

If your system seems to access the wrong disk, reset your computer. You may have selected some combination of options that are preventing the system from functioning properly.

Remember that when you specify a drive number, you are specifying a logical drive number which, based on your system's configuration, may point at drives in another order. If you have SYSGENed these settings, you will have to hold **CLEAR** down while you reset your computer.

Problem 2 . . . RS-232C communications do not work, or function incorrectly.

If you experience RS-232C problems, the first thing you should do is to make sure both "ends" are operating with the same RS-232C parameters (baud rate, word length, stop bits, and parity). If these parameters are not the same at each end, the data sent and received appears scrambled.

Some hardware, such as serial printers, require "handshaking" when running above a certain baud rate. It may be necessary to hook the hardware's handshake line (such as the BUSY line) to an appropriate RS-232C lead, such as CTS.

Problem 3 . . . Random system crashes, recurring disk I/O errors, system lock-up, and other random glitches keep happening.

If you encounter these types of problems, the first thing to check is the cable connections between the TRS-80 and the peripherals.

If you experience constant difficulty in disk read/write operations, it is possible that the disk drive heads need cleaning. There are kits available from Radio Shack to clean disk heads, or you may wish to have the disk drive serviced at a repair facility. If you need to frequently clean the disk heads, you might be using some defective disk media. Check the diskettes for any obvious signs of flaking or excess wear, and dispose of any that appear

even marginal. Tobacco smoke and other airborne contaminants can build up on disk heads, and can cause read/write problems. Disk drives in "dirty" locations may need to have their heads cleaned as often as once a week.

One common and often overlooked cause of random-type problems is static electricity. In areas of low humidity, static electricity is present, even if actual static discharges are not felt by the computer operator. Be aware that static discharges can cause system glitches, as well as physically damage computer hardware and disk media.

Error Messages

If the computer displays one of the messages listed in this appendix, an operating system error occurred. Any other error message refers to an application program error, and you should see your application program manual for an explanation.

When an error message is displayed:

- Try the operation several times.
- Look up operating system errors below and take any recommended actions. (See your application program manual for explanations of application program errors.)
- Try using other diskettes.
- Reset the computer and try the operation again.
- Check all the power connections.
- Check all interconnections.
- Remove all diskettes from drives, turn off the computer, wait 15 seconds, and turn it on again.
- If you try all these remedies and still get an error message, contact a Radio Shack Service Center.

NOTE: If there is more than one thing wrong, the computer might wait until you correct the first error before displaying the second error message.

This list of error messages is alphabetical, with the decimal and hexadecimal error numbers in parentheses. Following it is a quick reference list of the messages arranged in numerical order.

TRSDOS Error Messages

Attempted to read locked/deleted data record (Error 7, X'07')

Check for an error in your application program.

Attempted to read system data record (Error 6, X'06')

Check for an error in your application program.

Data record not found during read (Error 5, X'05')

Try the operation again. If it fails, use a different disk. Reformat the old disk; this should lock out the flaw. To retrieve your files from the flawed disk, perform a backup by class (see the BACKUP command). You may have to remove the files that have errors so BACKUP will work.

Data record not found during write (Error 13, X'0D')

Try the operation again. If it still fails, use a different disk.

Device in use (Error 39, X'27')

RESET the device in use before REMOVEing it.

Device not available (Error 8, X'08')

Make sure you enter the correct device specification and that the device peripheral is ready. You can use the DEVICE (B=ON) command to display all devices available to the system.

Directory full — can't extend file (Error 30, X'1E')

All directory slots are being used. Use BACKUP to copy some of the disk's files to a newly formatted disk.

Directory read error (Error 17, X'11')

Try the operation again, using a different drive. If it fails, use a different disk. You can try to get the files off a flawed disk by doing a backup by class. If this doesn't work, you must use your backup of the disk.

Directory write error (Error 18, X'12')

The directory may no longer be reliable. If the problem recurs, use a different diskette.

Disk space full (Error 27, X'1B')

While a file was being written, all available disk space was used. The file contains only the data written before the error occurred. Write the file to a disk that has more available space. Then, REMOVE the partial copy to recover disk space.

End of file encountered (Error 28, X'1C')

You tried to read past the end of file pointer. Use the DIR command to check the size of the file. Check for an error in your application program.

Extended error (Error 63)

An error has occurred and the extended error code is in the HL register pair.

File access denied (Error 25, X'19')

You specified a password for a file that is not password protected or you specified the wrong password for a file that is password protected.

File already open (Error 41, X'29')

Use the RESET library command to close the file before trying to open it.

File not in directory (Error 24, X'18')

Check the spelling of the filespec. Use the DIR command to see if the file is on the disk.

File not open (Error 38, X'26')

Open the file before trying to access it.

GAT read error (Error 20, X'14')

Try the operation again, using a different drive. If this fails, use a backup by class to move all files to a different disk. Then, try the operation again, using the new disk.

GAT write error (Error 21X'15')

The Granule Allocation Table may no longer be reliable. If the problem recurs, try the operation again, using a different drive. If it still fails, use a different disk.

HIT read error (Error 22, X'16')

Try the operation again, using a different drive. If this still fails, use a backup by class to copy the files to a different disk.

HIT write error (Error 23, X'17')

The Hash Index Table may no longer be reliable. If the problem recurs, try the operation again, using a different drive. If it still fails, use a different disk.

Illegal access attempted to protected file (Error 37, X'25')

The USER password was given for access to a file, but the requested access required the OWNER password. (See the ATTRIB command.)

Illegal drive number (Error 32, X'20')

The specified disk drive is not included in your system or is not ready for access (no diskette, non-TRSDOS Version 6 diskette, drive door open, and so on). See the DEVICE command.

Illegal file name (Error 19, X'13')

The specified filespec does not meet TRSDOS filespec requirements. See Chapter 1 for proper filespec syntax.

Illegal logical file number (Error 16, X'10')

Your program probably has altered the File Control Block improperly. Check for an error in your application program.

Load file format error (Error 34, X'22')

An attempt was made to load a file that cannot be loaded by TRSDOS. The file was probably a data file or a BASIC program file.

Lost data during read (Error 3, X'03')

Information was not transferred in the time allotted; therefore, it was lost. Try the operation again, using a different drive. If it still fails, use a different disk.

Lost data during write (Error 11, X'0B')

Information was not transferred in the time allotted; therefore, it was lost. Try the operation again, using a different drive. If it still fails, use a different disk.

LRL open fault (Error 42, X'2A')

The logical record length specified when the file was opened is different than the LRL used when the file was created. COPY the file to another file that has the specified LRL.

No device space available (Error 33, X'21')

You tried to SET a driver or filter and all of the Device Control Blocks were in use. Use the DEVICE command to see if any non-system devices can be removed to provide more space.

No directory space available (Error 26, X'1A')

You tried to open a new file and no space was left in the directory. Use a different disk or REMOVE some files you no longer need.

No error (Error 0)

The @ERROR supervisor call was called without any error condition being detected. A return code of zero indicates no error. Check for an error in your application program.

Parity error during header read (Error 1, X'01')

Try the operation again, using a different drive. If it still fails, use a different disk.

Parity error during header write (Error 9, X'09')

Try the operation again, using a different drive. If it still fails, use a different disk.

Parity error during read (Error 4, X'04')

Try the operation again, using a different drive. If it still fails, use a different disk.

Parity error during write (Error 12, X'0C')

Try the operation again, using a different drive. If it still fails, use a different disk.

Program not found (Error 31, X'1F')

Check the spelling of the filespec (you must include the /CMD extension). If this is not the problem, make sure the disk that contains the file is loaded.

Protected system device (Error 40, X'28')

You cannot REMOVE any of the following devices: *KI, *DO, *PR, *JL, *SI, *SO.

Record number out of range (Error 29, X'1D')

Correct the record number or try the operation again, using another copy of the file.

Seek error during read (Error 2, X'02')

Try the operation again, using a different drive. If it still fails, use a different disk.

Seek error during write (Error 10, X'0A')

Try the operation again, using a different drive. If it still fails, use a different disk.

— Unknown error code

The @ERROR supervisor call was called with an error number that is not defined. Check for an error in your application program.

Write fault on disk drive (Error 14, X'0E')

Try the operation again, using a different drive. If it still fails, use a different disk. If the problem continues, contact a Radio Shack Service Center.

Write protected disk (Error 15, X'0F')

Remove the write-protect tab, if the diskette has one. If it does not, use the DEVICE command to see if the drive is set as write protected. If it is, you can use the SYSTEM command with the (WP=OFF) parameter to write enable the drive. If the problem recurs, check the drive connections on the external drives, even if the error is occurring on an internal drive. Or, use a different drive or diskette.

TRSDOS ERROR MESSAGES

Decimal	Hex	Message
0	X'00'	No Error
1	X'01'	Parity error during header read
2	X'02'	Seek error during read
3	X'03'	Lost data during read
4	X'04'	Parity error during read
5	X'05'	Data record not found during read
6	X'06'	Attempted to read system data record
7	X'07'	Attempted to read locked/deleted data record
8	X'08'	Device not available
9	X'09'	Parity error during header write
10	X'0A'	Seek error during write
11	X'0B'	Lost data during write
12	X'0C'	Parity error during write
13	X'0D'	Data record not found during write
14	X'0E'	Write fault on disk drive
15	X'0F'	Write protected disk
16	X'10'	Illegal logical file number
17	X'11'	Directory read error
18	X'12'	Directory write error
19	X'13'	Illegal file name
20	X'14'	GAT read error
21	X'15'	GAT write error
22	X'16'	HIT read error
23	X'17'	HIT write error
24	X'18'	File not in directory
25	X'19'	File access denied
26	X'1A'	Full or write protected disk
27	X'1B'	Disk space full
28	X'1C'	End of file encountered
29	X'1D'	Record number out of range
30	X'1E'	Directory full — can't extend file
31	X'1F'	Program not found
32	X'20'	Illegal drive number
33	X'21'	No device space available
34	X'22'	Load file format error
37	X'25'	Illegal access attempted to protected file
38	X'26'	File not open
39	X'27'	Device in use
40	X'28'	Protected system device
41	X'29'	File already open
42	X'2A'	LRL open fault
43	X'2B'	SVC parameter error
63	X'3F'	Extended error
—		Unknown error code

BASIC ERROR CODES AND MESSAGES

Number	Message
1	NEXT without FOR A variable in a NEXT statement does not correspond to any previously executed FOR statement variable.
2	Syntax error BASIC encountered a line that contains an incorrect sequence of characters (such as unmatched parenthesis, misspelled statement, incorrect punctuation, etc.). BASIC automatically enters the edit mode at the line that caused the error.
3	RETURN without GOSUB BASIC encountered a RETURN statement for which there is no matching GOSUB statement.
4	Out of DATA BASIC encountered a READ statement, but no DATA statements with unread items remain in the program.
5	Illegal function call A parameter that is out of range was passed to a math or string function. An FC error may also occur as the result of: <ol style="list-style-type: none">A negative or unreasonably large subscript.A negative or zero argument with LOG.A negative argument to SQR.A negative mantissa with a noninteger exponent.A call to aUSR function for which the starting address has not yet been given.An improper argument to MID\$, LEFT\$, RIGHT\$, PEEK, POKE, TAB, SPC, STRING\$, SPACE\$, INSTR, or ON . . . GOTO.

-
- | | |
|----|--|
| 6 | Overflow
The result of a calculation was too large to be represented in BASIC numeric format. If underflow occurs, the result is zero and execution continues without an error. |
| 7 | Out of memory
A program is too large, or has too many FOR loops or GOSUBs, too many variables, or expressions that are too complicated. |
| 8 | Undefined line number
A nonexistent line was referenced in a GOTO, GOSUB, IF . . . THEN . . . ELSE, or DELETE statement. |
| 9 | Subscript out of range
An array element was referenced either with a subscript that is outside the dimensions of the array, or with the wrong number of subscripts. |
| 10 | Duplicate Definition
Two DIM statements were given for the same array, or a DIM statement was given for an array after the default dimension of 10 has been established for that array. |
| 11 | Division by zero
An expression includes division by zero, or the operation of involution results in zero being raised to a negative power. BASIC supplies machine infinity with the sign of the numerator as the result of the division, or it supplies positive machine infinity as the result of the involution. Execution then continues. |
| 12 | Illegal direct
A statement that is illegal in direct mode was entered as a direct mode command. |
| 13 | Type mismatch
A string variable name was assigned a numeric value or vice versa. A numeric function was given a string argument or vice versa. |
-

-
- 14** **Out of string space**
String variables have caused BASIC to exceed the amount of free memory remaining. BASIC allocates string space dynamically, until it runs out of memory.
- 15** **String too long**
An attempt was made to create a string more than 255 characters long.
- 16** **String formula too complex**
A string expression is too long or too complex. The expression should be broken into smaller expressions.
- 17** **Can't continue**
An attempt was made to continue a program that:
- a. Has halted due to an error.
 - b. Has been modified during a break in execution.
 - c. Does not exist.
- 18** **Undefined user function**
AUSR function was called before providing a function definition (DEF statement).
- 19** **No RESUME**
An error-handling routine was entered without a matching RESUME statement.
- 20** **RESUME without error**
A RESUME statement was encountered prior to an error-handling routine.
- 21** **Unprintable error**
An error message is not available for the error that occurred.
- 22** **Missing operand**
An expression contains an operator with no operand.
- 23** **Line buffer overflow**
An attempt was made to input a line with too many characters.
-

-
- 26 **FOR without NEXT**
A FOR statement was encountered without a matching NEXT.
- 29 **WHILE without WEND**
A WHILE statement does not have a matching WEND.
- 30 **WEND without WHILE**
A WEND statement was encountered without a matching WHILE.

Disk Errors

- 50 **FIELD overflow**
A FIELD statement is attempting to allocate more bytes than were specified for the record length of a direct-access file.
- 51 **Internal error**
An internal malfunction has occurred in BASIC. Report to Radio Shack the conditions under which the message appeared.
- 52 **Bad file number**
A statement or command references a file with a buffer number that is not OPEN or is out of the range of file numbers specified at initialization.
- 53 **File not found**
A LOAD, KILL, or OPEN statement references a file that does not exist on the current disk.
- 54 **Bad file mode**
An attempt was made to use PUT, GET, or LOF with a sequential file, to LOAD a direct file, or to execute an OPEN statement with a file mode other than I, O, R, E or D.
- 55 **File already open**
An OPEN statement for sequential output was issued for a file that is already open; or a KILL statement was given for a file that is open.

-
- | | | |
|----|---------------------------------|---|
| 57 | Device I/O error | An Input/Output error occurred. This is a fatal error; the operating system cannot recover from it. |
| 58 | File already exists | The filespec specified in a NAME statement is identical to a filespec already in use on the disk. |
| 61 | Disk full | All disk storage space is in use. |
| 62 | Input past end | An INPUT statement was executed after all the data in the file had been INPUT, or for a null (empty) file. To avoid this error, use the EOF function to detect the end-of-file. |
| 63 | Bad record number | In a PUT or GET statement, the record number is either greater than the maximum allowed (65,535) or equal to zero. |
| 64 | Bad file name | An illegal filespec (file name) was used with a LOAD, SAVE, KILL, or OPEN statement (for example, a filespec with too many characters). |
| 66 | Direct statement in file | A direct statement was encountered while LOADING an ASCII-format file. The LOAD is terminated. |
| 67 | Too many files | An attempt was made to create a new file (using SAVE or OPEN) when all directory entries are full. |
| 68 | Disk write protected | |
| 69 | File access denied | |
| 70 | Command Aborted | |

Appendix E/ Converting TRSDOS Version 1 BASIC Programs to TRSDOS Version 6 BASIC Programs

You can run a TRSDOS Version 1 applications program on TRSDOS Version 6. However, you may need to make a few changes to the program. The differences between the two BASICs are listed below. (From here on, we will refer to TRSDOS Version 6 as TRSDOS 6, and to TRSDOS Version 1 as TRSDOS 1).

1. **ROM Subroutines.** TRSDOS 1 BASIC is a ROM and RAM-based language. TRSDOS 6 BASIC is strictly a RAM language; therefore, it cannot access any of TRSDOS 1's ROM subroutines.
2. **Disk Files.** TRSDOS 6 BASIC does not provide cassette support. It is exclusively a "disk system", that is, you can only use it with floppy diskettes or with a hard disk system. If you have learned BASIC through "Getting Started with TRS-80 BASIC", or have never worked with a disk system before, read about "Disk Files" in Chapter 5. This chapter explains how you can store and access data on disk. You also need to read Chapter 1, "Sample Session", which describes how to load disk BASIC and how to save a program on disk.
3. **Characters per Line.** Both TRSDOS 1 and TRSDOS 6 BASIC allow you to type up to 255 characters per line. However, there is a slight difference. With TRSDOS 6, you can type up to 249 characters per line. The other six characters are reserved for the line number and the space following the line number. With TRSDOS 1, you can type up to 240 characters in the command mode, and add the extra 15 characters in the edit mode.
4. **Variable Names.** TRSDOS 1 BASIC only recognizes the first two letters of a variable name; TRSDOS 6 BASIC allows variable names of up to 40 characters, all of which are significant.
5. **Converting to Integers.** In converting a single or double-precision number to integer value, TRSDOS 1 BASIC truncates the number; TRSDOS 6 BASIC rounds the number. This difference in conversions also affects assignment statements and function or statement evaluations. For example, if you typed $I\% = 2.5$, TRSDOS 1 BASIC would convert 2.5 to 2; TRSDOS 6 BASIC would convert it to 3. If you typed `TAB(4.5)`, TRSDOS 1 would move to the fourth tab position; TRSDOS 6 would move to the fifth tab position.

If you enter a number as a constant in response to a command that calls for an integer, and the number is out of integer range, BASIC converts it to single or double precision. When the number is printed, it appears with a type-declaration tag at the end.

6. **Print Zones.** TRSDOS 1 BASIC includes 16 spaces between print zones; TRSDOS 6 BASIC includes 20 spaces. This is because TRSDOS 1's screen displays up to 64 characters horizontally,

while TRSDOS 6's screen displays up to 80 characters horizontally.

7. **Ports.** If your program uses PEEKs or POKEs, it is probably accessing Model III ports. The Model 4 and Model 4P assigned ports are different. For information about these ports, refer to the Technical Reference Manual.
8. **BASIC Keywords.** The following TRSDOS 1 BASIC keywords are not supported by TRSDOS 6 BASIC: CSAVE, CLOAD, POINT, CLOCK, CMD, POSN, RENAME, and VERIFY.

The following TRSDOS 6 BASIC keywords are not supported by TRSDOS 1 BASIC: COMMON, ERR\$, OCT\$, OPTION BASE, RENUM, ROW, SPACE\$, SPC, SWAP, WAIT, WHILE . . . WEND, WIDTH, and WRITE#.
9. **Reserved Words.** TRSDOS 6 BASIC **requires that all reserved words be delimited by spaces.** Only those characters which may be part of the keyword's syntax can be typed immediately after or before the keyword. For all other characters, leave a space between the keyword and the character. (For example, you cannot type DEFUSR; you must leave a space between DEF and USR.) Appendix F includes a listing of Reserved Words.
10. **Error Messages.** TRSDOS 6 BASIC Error Codes, Character Codes and Internal Codes for BASIC keywords differ from TRSDOS 1 BASIC codes. See the Appendices for more information on TRSDOS 6 BASIC codes.
11. **String Space.** TRSDOS 6 BASIC allocates string space dynamically; you do not need to allocate string space with the CLEAR statement. Instead, use CLEAR to set the maximum memory location BASIC may access and the amount of stack space. For more information, see CLEAR in Chapter 7.
12. **Printing Single and Double-Precision Numbers.** The rules for printing single and double-precision numbers are different. For more information, see PRINT in Chapter 7.
13. **Division by Zero.** Contrary to TRSDOS 1 BASIC, TRSDOS 6 BASIC does not produce a fatal error if it encounters division by zero or overflow. Instead, it prints an error message and continues executing your program.
14. **FOR . . . NEXT.** TRSDOS 6 BASIC skips the body of a FOR . . . NEXT loop if the initial value of the loop, times the sign of the STEP, exceeds the final value of the loop, times the sign of the STEP. For a more detailed explanation, see FOR . . . NEXT in Chapter 7.
15. **Nested Subroutines.** If your program has nested subroutines or nested FOR . . . NEXT loops, an "Out of memory" error may

occur. To avoid this, use the CLEAR statement to set aside "stack space" for your subroutines. See CLEAR in Chapter 7 for more information.

16. **IF . . . THEN . . . or IF THEN . . . ELSE.** With TRSDOS 1 BASIC, the word "THEN" is optional in both of these statements. With TRSDOS 6 BASIC, it is required.
17. **PRINT@ and PRINT TAB.** If a string is too long to fit on the current line, TRSDOS 6 BASIC prints the entire string on the next line. TRSDOS 1 BASIC prints as many characters as possible on the first line, and the rest on the second line.

In a PRINT TAB(*n*) statement, if *n* is greater than 80, TRSDOS 6 BASIC divides *n* by 80. The remainder of this division is used as the tab position. For example, if you typed TAB(91), TRSDOS 6 BASIC would tab to position 11 on the screen. TRSDOS 1 BASIC would tab to position 91.

18. **Self-Documenting Programs.** TRSDOS Version 6 BASIC programs can be self-documenting, as in the following example:

```
100 INPUT EFFORT
110 INPUT DISTANCE
120 FORCE = EFFORT * DISTANCE
130 PRINT FORCE
140 END
```

Under TRSDOS Version 6 BASIC, the reserved words (FOR and TAN in the above example) in the variable names will not cause a syntax error. This is because in order to be recognized as reserved words, they must be delimited by surrounding spaces.

TRSDOS Version 1 would return syntax errors in this example.

19. **Graphics Characters.** Under TRSDOS Version 6, the size of the graphics characters is different than under TRSDOS Version 1. The lowest portions of the TRSDOS Version 6 graphics characters are smaller than their TRSDOS Version 1 equivalents.

Appendix F/ BASIC Keywords and Derived Functions

Reserved BASIC Words

ABS	FN	OR	VAL
AND	FRE	OUT	VARPTR
ASC	GET	PEEK	WAIT
ATN	GOSUB	POKE	WEND
AUTO	GOTO	POS	WHILE
CALL	HEX\$	PRINT	
CDBL	IF	PUT	WRITE
CHAIN	IMP	RANDOM	XOR
CHR\$	INKEY\$	READ	+
CLEAR	INP	REM	-
CLOSE	INPUT	RENUM	*
CLS	INSTR	RESTORE	/
COMMON	INT	RESUME	^
CONT	KILL	RETURN	\
COS	LEFT\$	RIGHT\$,
CSNG	LEN	RND	>
CVD	LET	ROW	=
CVI	LINE	RSET	<
CVS	LIST	RUN	
DATA	LLIST	SAVE	
DATE\$	LOAD	SGN	
DEF	LOC	SIN	
DEFDBL	LOF	SOUND	
DEFINT	LOG	SPACE\$	
DEFSNG	LPOS	SPC	
DEFSTR	LPRINT	SQR	
DELETE	LSET	STEP	
DIM	MEM	STOP	
EDIT	MERGE	STR\$	
ELSE	MID\$	STRING\$	
END	MKD\$	SWAP	
EOF	MKI\$	SYSTEM	
EQV	MKS\$	TAB	
ERASE	MOD	TAN	
ERL	NAME	THEN	
ERR	NEW	TIMES	
ERROR	NEXT	TO	
ERRS\$	NOT	TROFF	
EXP	OCT\$	TRON	
FIELD	ON	USING	
FIX	OPTION	USR	

Internal Codes for BASIC Keywords

ABS	65414	GOTO	137
AND	248	HEX\$	65434
ASC	65429	IF	139
ATN	65422	IMP	252
AUTO	171	INKEY\$	224
CALL	182	INP	65424
CDBL	65438	INPUT	133
CHAIN	185	INSTR	219
CHR\$	65430	INT	65413
CINT	65436	KILL	200
CLEAR	146	LEFT\$	65409
CLOSE	195	LEN	65426
CLS	159	LET	136
COMMON	184	LINE	177
CONT	153	LIST	147
COS	65420	LLIST	158
CSNG	65437	LOAD	196
CVD	65452	LOC	65454
CVI	65450	LOF	65455
CVS	65451	LOG	65418
DATA	132	LPOS	65435
DATE\$	222	LPRINT	157
DEF	151	LSET	201
DEFDBL	176	MEM	225
DEFINT	174	MERGE	197
DEFSNG	175	MID\$	65411
DEFSTR	173	MKD\$	65458
DELETE	170	MKI\$	65456
DIM	134	MKS\$	65457
EDIT	167	MOD	253
ELSE	162	NAME	199
END	129	NEW	148
EOF	65453	NEXT	131
EQV	251	NOT	214
ERASE	166	OCT\$	65433
ERL	215	ON	149
ERR	216	OPEN	191
ERROR	168	OPTION	186
ERRS\$	223	OR	249
EXP	65419	OUT	156
FIELD	192	PEEK	65431
FIX	65439	POKE	152
FN	212	POS	65425
FOR	130	PRINT	145
FRE	65423	PUT	194
GET	193	RANDOM	187
GOSUB	141	READ	135

REM	143	TIMES\$	226
RENUM	172	TO	207
RESTORE	140	TROFF	164
RESUME	169	TRON	163
RETURN	142	USING	218
RIGHT\$	65410	USR	211
RND	65416	VAL	65428
ROW	65459	VARPTR	221
RSET	202	WAIT	150
RUN	138	WEND	181
SAVE	203	WHILE	180
SGN	65412	WIDTH	161
SIN	65417	WRITE	183
SOUND	205	XOR	250
SPACES\$	65432	+	243
SPC	213	-	244
SQR	65415	*	245
STEP	210	/	246
STOP	144	^	247
STR\$	65427	\	254
STRING\$	217	'	220
SWAP	165	>	240
SYSTEM	189	=	241
TAB	209	<	242
TAN	65421		
THEN	208		

Derived BASIC Functions

Functions which are not intrinsic to BASIC may be calculated as follows:

Function	BASIC Equivalent
SECANT	$SEC(X) = 1/COS(X)$
COSECANT	$CSC(X) = 1/SIN(X)$
COTANGENT	$COT(X) = 1/TAN(X)$
INVERSE SINE	$ARCSIN(X) = ATN(X/SQR(-X*X + 1))$
INVERSE COSINE	$ARCCOS(X) = ATN(X/SQR(-X*X + 1)) + 1.5708$
INVERSE SECANT	$ARSCEC(X) = ATN(X/SQR(X*X - 1)) + SGN(SGN(X) - 1)*1.5708$
INVERSE COSECANT	$ARCCSC(X) = ATN(X/SQR(X*X - 1)) + (SGN(X) - 1)*1.5708$
INVERSE COTANGENT	$ARCCOT(X) = ATN(X) + 1.5708$
HYPERBOLIC SINE	$SINH(X) = (EXP(X) - EXP(-X))/2$
HYPERBOLIC COSINE	$COSH(X) = (EXP(X) + EXP(-X))/2$
HYPERBOLIC TANGENT	$TANH(X) = (EXP(-X)/EXP(X) + EXP(-X))*2 + 1$
HYPERBOLIC SECANT	$SECH(X) = 2/(EXP(X) + EXP(-X))$
HYPERBOLIC COSECANT	$CSCH(X) = 2/(EXP(X) - EXP(-X))$
HYPERBOLIC COTANGENT	$COTH(X) = (EXP(-X)/(EXP(X) - EXP(-X))*2 + 1$
INVERSE HYPERBOLIC SINE	$ARCSINH(X) = LOG(X + SQR(X*X + 1))$
INVERSE HYPERBOLIC COSINE	$ARCCOSH(X) = LOG(X + SQR(X*X - 1))$
INVERSE HYPERBOLIC TANGENT	$ARCTANH(X) = LOG((1 + X)/(1 - X))/2$
INVERSE HYPERBOLIC SECANT	$ARCSECH(X) = LOG((SQR(-X*X + 1) + 1)/X)$
INVERSE HYPERBOLIC COSECANT	$ARCCSCH(X) = LOG((SGN(X)*SQR(X*X + 1) = 1)/X)$
INVERSE HYPERBOLIC COTANGENT	$ARCCOTH(X) = LOG((X + 1)/(X - 1))/2$

Appendix G/ Video Display Worksheet

0	79
1	159
2	238
3	318
4	398
5	479
6	559
7	639
8	719
9	799
10	879
11	959
12	1039
13	1119
14	1199
15	1279
16	1359
17	1439
18	1519
19	1599
20	1679
21	1759
22	1839
23	1919

0	0
80	1
160	2
240	3
320	4
400	5
480	6
560	7
640	8
720	9
800	10
880	11
960	12
1040	13
1120	14
1200	15
1280	16
1360	17
1440	18
1520	19
1600	20
1680	21
1760	22
1840	23

Appendix H/ Glossary

alphanumeric — consisting of only the letters A-Z, a-z, and the numerals 0-9.

ASCII — The alphanumeric representation of controls and characters as a single byte, falling within a range from 1 to 127 (sometimes including 0).

ASCII files — Files that are readable by LISTing the file. Source, text, and data files are usually ASCII files.

background task — A job performed by the computer that is not apparent to the user or does not require interaction with the user. Some examples are the REAL TIME CLOCK, the SPOOLer, and the TRACE function.

baud — Refers to the rate of serial data transfer.

bit — One eighth of a byte; one binary digit.

boot — The process of resetting your computer and loading in the resident operating system from the system drive.

buffer — An area in RAM that temporarily holds information that is being passed between devices or programs.

byte — The unit that represents one character to the Model 4. It is composed of eight binary "bits" that are either ON (1) or OFF (0). One byte can represent a number from 0 to 255.

COMM — A communications program capable of interacting with: disk, printer, video display, keyboard, and the RS232 interface. COMM dynamically buffers all of the system devices.

concatenate — To add one variable or string onto the end of another.

configuration — The status of the system and physical devices that are available to it. This configuration can be dynamically changed with several library commands, and can be saved with the SYSGEN library command. If the system is SYSGENed, the SYSGENed configuration is re-established each time the machine is reset or re-started.

cursor — The location on the video display where the next character is printed. It is marked by the presence of a cursor character.

cylinder — All tracks of the same number on a disk drive. On single sided drives, cylinders are the same as tracks.

DAM (Data Address Mark) — A control byte that prefixes each sector on a disk. This byte indicates the type of sector that is about to be read. It can mark a sector as being deleted or undeleted, a user sector or a system sector.

DCB — Device Control Block, a small piece of memory used to control the status, input, and output of data between the system and the devices.

DCT — Drive Code Table, a piece of memory containing information about the disk drives and/or diskettes in them.

density — Refers to the density of the data written to a diskette. Double density provides approximately 80% more capacity than single density.

device — The two types of devices are Logical and Physical.

A logical device is one that is referred to in TRSDOS. Logical devices have devspecs, a 2-character name that is prefixed with an asterisk (*). An example of a logical device is *PR, which is normally used to send data to the printer.

A physical device is a piece of hardware, such as the video display or printer. A piece of software called a "driver" connects the logical device to the physical device by translating data from the format used by logical devices into the format required by the hardware, and vice versa.

devspec — The name associated with a device by which it is referenced. A devspec always consists of three characters: an asterisk followed by two alphabetic characters.

directory — An area of a disk that contains the names of the files on the disk, information on where the data in those files is stored on the disk, and other information such as any password, the logical record length, the modification date, and so on.

disk I.D. — A disk's name and master password assigned when it is formatted.

***DO** — The Video Display device.

:drive — Indicate that a drive number can be inserted where this is used. A drive number must always be preceded immediately by a ":".

driver — A program that interfaces a physical device (a piece of hardware) to a logical device, which can be referenced by TRSDOS.

EOF — End of File, a marker used to denote the end of a program or data file.

/ext — The extension of a filespec. The use of /ext is sometimes optional. An extension's first character must be a "/" (slash) which is followed by one to three alphanumeric characters, the first of which must be a letter.

FCB — File Control Block, a small piece of memory used to control the status and I/O of data between the operating system and disk files.

filename — The mandatory name used to reference a disk file. A filename consists of one to eight alphanumeric characters, the first of which must be alphabetic.

filespec — A disk file's name. A filespec consists of four fields and two switches. The first field is always mandatory. A filespec is in the following format:

!filename/ext.password:drive!

“!” — (preceding filename) is an optional switch. If you specify this switch, you can build a file with the same name as a TRSDOS command or utility. For example, you can issue the command: LIST !DEVICE and TRSDOS will list the user-created file named DEVICE.

filename — The mandatory name of the file.

/ext — The optional file extension.

.password — The optional file password.

:drive — The optional drive number.

“!” — (following :d) is an optional switch. If this switch is set, the end of file marker for filespec is updated after every write to the file.

filter — A machine language program that monitors and/or alters I/O that passes through it. FILTER is also the library command that establishes a FILTER routine.

/FIX — The desired file extension for a PATCH file.

foreground task — Jobs performed by the computer that are apparent to the user, such as running an applications program.

gran — The abbreviation of granule. A gran is the minimum amount of storage used for a disk file. As files are extended, file allocation is increased in increments of grans. The size of a gran varies with the size and density of a diskette.

HIGH\$ — The name of a memory location in the operating system that contains the address of the highest unprotected memory address available for use. Programs that are above this location are protected from other programs. You can display or change the value of HIGH\$ by using the MEMORY command or the @HIGH\$ SVC.

interrupt — A signal generated by the hardware which causes the system to stop what it is doing to perform some other service. These interruptions are used to perform background tasks such

as checking the keyboard for input and supplying data to the printer if the spooler is running.

I/O — The abbreviation for Input/Output.

/JCL — The desired file extension for a DO file. JCL is the abbreviation for Job Control Language.

***JL** — The Joblog device.

***KI** — The Keyboard device.

/KSM — The desired file extension for a ksm file. KSM is an abbreviation for Key-Stroke Multiply.

library — A set of commands that perform most of the operating system functions.

load module format — A file format that loads directly to a specified RAM address.

LSB — The Least Significant Byte. In a hexadecimal word, it is sometimes referred to as the “low order byte”.

macro — Predetermined lines of code used in JCL.

mod date — The date a file was last written to.

mod flag — A “+” sign placed after a filename that indicates it was written to since its last backup.

MSB — The Most Significant Byte. In a hexadecimal word, it is sometimes referred to as the “high order byte”.

NIL — A “dummy” device which a logical device can be linked or routed to. When you reset a user-defined device, it points at NIL. NIL discards any data that is sent to it and returns a null (ASCII 0) when data is requested from it. It is useful when you want to discard output from a program during a test run.

NRN — Next Record Number.

parameter — an optional value that you supply to a command line. Parameters may follow a command or utility and are enclosed in parentheses ().

parse — The process of breaking a command into individual parameters.

partspec — A way to represent a group of one or more files by entering only part of the file specification. Partspecs are allowed in some TRSDOS library and utility commands so that a group of files can be specified. A partspec can consist of any combination of the four fields that make up a filespec.

In a partspec, a dollar sign (\$) can be used to represent any character in a given position in a filespec. This is called “wildcarding.”

By prefixing a partspec with a minus sign (–), you can cause all the files *except* those that match the partspec to be considered in the given command.

.password — The optional password associated with a filespec. A password's first character is a period (.) and it is followed by one to eight alphanumeric characters, the first of which must be a letter.

PATCH — A utility that makes minor alterations to disk files.

***PR** — The Line Printer device.

RAM — Random Access Memory. This type of memory can be accessed in any order, and any byte can be read or written at any time.

ROM — Read Only Memory. This type of memory stores information that will not change. ROM does not require power to maintain its data.

sector — A contiguous 256-byte block of disk storage. Each sector has an I.D. field which contains its track and sector number. This allows the hardware to use the proper area of the disk when reading or writing to the disk.

A sector is the smallest amount of data the operating system will read from or write to a disk. Several sectors make up a track. One or more tracks make up a cylinder.

***SI** — The Standard Input device. Programs that read data from this device normally receive data from the keyboard. You can change this to have data read from a file or another device by issuing a ROUTE command. This allows a program to accept input from any device without the need to modify the program.

***SO** — The Standard Output Device. Data that is output to this device by a program is normally displayed on the screen. You can change this to have the data written to a file or another device by issuing a ROUTE command. This allows a program to perform output to any device without the need to change the program.

switch — A parameter with a definite setting, such as ON/OFF or YES/NO.

token — A variable used in JCL.

utility — A program that provides a service to the user. Utilities differ from library commands as they are usually larger programs and require memory that is usually reserved for the user.

word — A 16-bit value which is stored in two contiguous 8-bit bytes. A word may be specified in hexadecimal format X'nnnn' or in decimal format nnnnn where nnnnn is a value from 0 to 65535.

Appendix I/TRSDOS Programs

This appendix contains five TRSDOS programs that you can use with the SET, SYSTEM, and FILTER library commands. There is a short explanation and examples for each program.

JOBLOG

```
ROUTE *JL [TO] filespec
ROUTE *JL [TO] devspec
```

Establishes the TRSDOS Joblog device (*JL), which collects certain information and sends it to a *filespec* or *devspec*.

You can use JOBLOG to create a file that contains a list of commands that you issue.

The information sent to *filespec* or *devspec* consists of all commands entered or received and the time (according to the system clock) that the commands occur.

When you issue a RESET *JL command, the Joblog function ceases and *filespec* closes. See the ROUTE library command for additional information.

To view the contents of a Joblog file, issue a RESET *JL command to close the file, and then a LIST command to list the file's contents.

To view the contents of a Joblog disk file when it is open, add a "trailing exclamation point" (!) to *filespec* (see "filespec" in the GLOSSARY). Then use the LIST library command to list the file to the screen or printer.

NOTE: If *filespec* already exists, information sent to it is appended to the end of the file.

Examples

```
ROUTE *JL TO LISTER/JBL ENTER
```

sends a log of all commands entered and received to the file LISTER/JBL.

```
ROUTE *JL TO *PR ENTER
```

sends a log of all commands entered and received to the printer.

KSM/FLT

<pre>SET <i>devspec</i> KSM/FLT [USING] <i>filespec</i> [(<i>parameter</i>)] FILTER *KI <i>devspec</i></pre>	Filter
--	---------------

Establishes the KSM (Key Stroke Multiply) filter.

You can use KSM/FLT to assign repetitive tasks (such as issuing a TRSDOS command) to one key, so that you only have to press **CLEAR** and the assigned key to execute the task.

devspec is any user-created *devspec*.

filespec contains up to 26 "key equivalents." The KSM filter loads the key equivalents from *filespec* into high memory.

The *parameter* is:

ENTER = *value* specifies *value* as the character TRSDOS recognizes as an **ENTER** character in a KSM file. *value* is a number in the hexadecimal format X'nn', a decimal number, or a single character such as a colon (:).

Each key equivalent is associated with the **CLEAR** key and an alphabetic key. When you press **CLEAR** and a key, TRSDOS executes the phrase associated with that key.

Building a KSM File

You can use the BUILD library command to build a /KSM file. To build a KSM file named ROUTINE/KSM, type:

```
BUILD ROUTINE/KSM ENTER
```

TRSDOS then lets you enter up to 26 key equivalents with the prompts:

```
A =>  
B =>  
C =>  
. .  
Z =>
```

To assign a character, type in the desired command; then terminate the line by pressing **ENTER** (or the character you specified with the ENTER parameter). To skip a character, press **ENTER** at the prompt. Pressing **ENTER** does not place an **ENTER** character at the end of the key equivalent, but merely terminates your input for that key. To place an **ENTER** character in a key equivalent, type a semicolon (;) where you wish an **ENTER** press to be executed. Each line can be up to 255 characters long.

When you have assigned all 26 characters, the file is closed and the BUILD terminates. Pressing **CONTROL(SHIFT)@** terminates the BUILD at any time.

If you want to create characters or strings that are not available from the keyboard, use the (HEX) parameter of the BUILD command.

It is not absolutely necessary to use the BUILD command with the /KSM extension to create a KSM file. The KSM/FLT program can use any file in ASCII format. TRSDOS uses the same rules concerning **ENTER** and the semicolon for a file in an ASCII format.

If you wish to deactivate the KSM filter, issue the command:

```
RESET *KI ENTER
```

If you wish to change to a different KSM file, issue the commands:

```
RESET *KI ENTER  
RESET devspec ENTER
```

And re-issue the commands for the new file:

```
SET devspec KSM/FLT [USING] filespec ENTER  
FILTER *KI devspec ENTER
```

Examples

```
A=>DIR :0 ENTER
```

specifies the key equivalent of A as "DIR :0". The command DIR :0 is displayed on the screen when the **CLEAR** and **A** keys are pressed together. The command is not executed until you press the **ENTER** key.

```
B=>FREE ; ENTER
```

specifies the key equivalent of B as "FREE;". A semicolon in a key equivalent represents an **ENTER** character. So, when you press **CLEAR** and **B**, the FREE library command is executed immediately (since the last character of the phrase is a semicolon).

```
F=>FREE ;DEVICE ; ENTER
```

specifies the key equivalent of F as "FREE;DEVICE;". A semicolon in a key equivalent represents an **ENTER** character. So, when you press

CLEAR and **B**, the FREE and DEVICE library commands are executed immediately.

Error Conditions

Attempting to SET a device to the KSM/FLT when it is already active on another device results in an error.

When you install an additional KSM, the new KSM file cannot be larger than the first KSM file installed.

COM/DVR

SET *CL TO COM/DVR	Driver
---------------------------	---------------

In order to use the Communications Line device (*CL), you must SET it to this driver program.

You can use COM/DVR to prepare the Communications Line (*CL) for use.

COM/DVR sets *CL to the RS-232C hardware.

After you SET *CL to the RS-232C hardware, you can alter the parameters of the RS-232C port with the SETCOM command.

Example

```
SET *CL TO COM/DVR
```

sets *CL to its driver program.

```
SETCOM (WORD=8,PARITY=OFF) ENTER
```

configures the RS-232C port using the values specified.

Technical Information

When you set the COM/DVR, it will be placed in high memory if there is not enough room in low memory. (Low memory is within TRSDOS and does not take away from the memory available for your programs.) If this happens, a message similar to the following appears:

Note: driver installed in high memory.

If you want to use Memdisk while you are using COM/DVR, be sure to install Memdisk first.

FORMS/FLT

<pre>SET *FF TO FORMS/FLT FILTER *PR *FF</pre>	Filter
--	---------------

You can use FORMS/FLT to prepare the printer filter (*FF) for use.

In order to use the printer filter (*FF), you must SET it to this filter program, and activate it with the FILTER command.

After you SET *FF to FORMS/FLT, you can set up the parameters of the printer filter with the FORMS command.

Example

```
SET *FF TO FORMS/FLT (ENTER)
```

sets *FF to its filter program.

```
FILTER *PR *FF
```

filters the printer to the printer filter program.

```
FORMS (MARGIN=12,CHARS=70,INDENT=17) (ENTER)
```

configures the printer filter by causing all lines to start 12 spaces in from the normal left-hand starting position. Any line longer than 70 characters is indented 17 spaces (5 spaces past the margin) when wrapped around, so it is printed starting at position 7.

Technical Information

When you set the FORMS/FLT, it will be placed in high memory if there is not enough room in low memory. (Low memory is within TRSDOS and does not take away from the memory available for your programs.) If this happens, a message similar to the following appears:

Note: filter installed in high memory.

If you want to use Memdisk while you are using FORMS/FLT, be sure to install Memdisk first.

MEMDISK/DCT

SYSTEM (DRIVE = <i>drive</i>, DRIVER = "MEMDISK")	Driver
--	---------------

Lets you add a pseudo floppy drive to the system which keeps its files in memory. Files stored on this drive can be accessed, read, and written more rapidly than files on a floppy. Only one Memdisk can be installed at a time.

All TRSDOS utilities treat the Memdisk drive as any other drive, so you can COPY, BACKUP, REMOVE, PURGE, ATTRIB, and display the DIRectory of the files on the Memdisk.

drive is the drive number you wish Memdisk to be. If you specify a drive number that is already defined, it is disabled and the Memdisk takes its place. *drive* is a number from 1 to 7.

To Install the Memdisk

When you start Memdisk, the following menu is displayed:

```
<A> Bank 0 (Primary Memory)
<B> Bank 1
<C> Bank 2
<D> Banks 1 and 2
<E> Disable MemDISK
```

```
Which type of allocation -
<A>, <B>, <C>, <D>, or <E>?
```

Each bank contains 32K of memory. If your system has only 64K of memory, then you do not have Banks 1 and 2.

Bank 0 is the top half of user memory. (See the Memory Map in the Technical Reference Manual.) It is shared by programs, drivers, filters, and Memdisk.

Because it is shared, if you select Bank 0 you are prompted for the number of cylinders that are to be used for the Memdisk in Bank 0. Selecting the number of cylinders allows you to use Memdisk but still have enough memory for the programs you want to run. You must select at least 3 cylinders. If you format Memdisk, the amount of memory used by each cylinder is shown below:

Double Density $256 \times 18 = 4608$ bytes per cylinder (4.5K)
Single Density $256 \times 10 = 2560$ bytes per cylinder (2.5K)

If you specify Banks 1 or 2, then all of the bank (32K) is used. If you specify option **(D)**, then Memdisk uses Banks 1 and 2 (64K).

After selecting which bank you want to use, you see:

```
Single or Double Density <S,D>?
```

This allows you to adjust the way memory is formatted. You get the same amount of space at single density as you do at double density, but the number of sectors per cylinder differs.

This feature allows mirror image backups to be performed, which allows data to be loaded into and out of Memdisk much faster.

Memdisk looks exactly like a floppy disk to a program.

If you selected Bank 0 (Double Density), the following message is displayed:

```
Note: Each Cylinder equals 4.50K of space.  
Number of free cylinders 1-N ?
```

N can be from 1 to 12. The value of N varies according to the number of other drivers resident in memory.

If you specified Bank 0 (Single Density), the following message is displayed:

```
Note: Each Cylinder equals 2.50K of space.  
Number of free cylinders 3-N ?
```

Enter the number of cylinders you want Memdisk to use in Bank 0, using the formula on the previous page. N can be from 3 to 7.

After you enter the configuring information, the following prompt is displayed:

```
Do you wish to Format it <Y,N>?
```

If you have not used Memdisk before, press **(Y)**. Formatting is not optional upon initial installation. MEMDISK is not initially installed unless you format it.

If you have used Memdisk and the system failed for some reason, press **(N)** to retrieve files that were left in Memdisk when it was last used. Remember that if the power went off, the Memdisk contents were erased.

If you answered the format question with **(Y)**, you see the message:

```
Verifying RAM Cylinder NN
```

```
Verifying Complete, RAM good  
Directory has been placed on Cylinder 1
```

MemDISK Successfully Installed

At this point, the Memdisk has been added to your system. The disk name is MEMDISK. It can be treated just like a floppy disk drive until you disable it or you reset the system.

To Disable the Memdisk

If you want to disable the Memdisk, then you must issue the command:

```
SYSTEM (DRIVE=drive,DRIVER="MEMDISK") (ENTER)
```

Then, at the menu select the **(E)** option. Memdisk displays one of the following messages:

```
MemDISK disabled, memory now available  
MemDISK disabled, Unable to reclaim high memory  
MemDISK disabled, Unable to reclaim driver area  
MemDISK disabled, Unable to reclaim high memory  
and driver area
```

If you receive the first message, Memdisk was disabled and was able to reclaim all memory (driver area, high memory (Bank 0), and alternate memory banks 1 and 2) that it was using.

If you receive the second message, Memdisk was unable to reclaim high memory (Bank 0) because another driver or filter was installed after Memdisk was set up and the other program is still in the way. This is known as memory fragmentation. If you need to use this area of memory, then you must reset the system.

If you receive the third message, Memdisk was disabled and able to claim high memory or alternate bank memory, but it could not reclaim the driver area.

If you receive the fourth message, Memdisk was disabled, but it could not reclaim any memory.

Error Conditions

Memdisk should be installed *before* COM/DVR or FORMS/FLT are. Filters and drivers can be loaded into an area within TRSDOS called low memory. (This area does not take away from the memory available for your programs.) However, not all of the drivers and filters can fit into this area at the same time. If there is no room left in low memory, most of the drivers and filters can be loaded in high memory. Since low memory works on a first come, first served basis and Memdisk is the only driver or filter that *must* load into low memory, you should install Memdisk before the other drivers and filters. This ensures that there is space available in low memory for Memdisk to reside.

If you attempt to re-install Memdisk in a different area of memory than the area that it was originally installed in, you get the error message "MemDISK already Active." Memdisk must always be re-installed as it was initially installed.

If you specify the wrong drive number (in the SYSTEM (DRIVE = *drive*, DRIVER = "MEMDISK") command) and you attempt to disable the Memdisk, then you receive the error message "Target Drive not a MemDISK."

If you attempt to disable a Memdisk and there is no MemDISK in the system to disable, then you receive the error message "MemDISK not present."

Technical Information

A Bank 0 Memdisk and BASIC use the same area of memory (RAM). Since a Bank 0 Memdisk and BASIC use the same area of memory, we recommend that you do not use BASIC when Memdisk is resident in Bank 0.

If you are going to use Memdisk as the system drive, you must COPY SYS0/SYS to it before it becomes the system drive. After Memdisk becomes the system drive, you can REMOVE SYS0/SYS from the Memdisk.

FLOPPY/DCT

Driver

FLOPPY/DCT is used in Radio Shack hard disk installations. See your hard disk manual for an explanation on how to use this driver.

CLICK/FLT

<pre>SET <i>devspec</i> CLICK/FLT FILTER *KI <i>devspec</i></pre>	Filter
---	---------------

You can use the key-click filter to produce a tone from the sound generator inside your computer.

In order to use the click filter, you must SET it to this filter program, and activate it with the FILTER command.

After you have set and activated the click filter, each time you press a key on the keyboard, your computer produces a tone. You can use this tone as an "auditory feedback".

You can change the pitch and duration of the tone produced by the key-click filter. To do this, you must apply a patch to the values that control the pitch and duration of the sound in CLICK/FLT.FILTER (see the PATCH command).

The patch is:

```
D00,7E = xx,yy;F00,7E = 03,00
```

xx is the duration of the tone. *xx* can be 1 - FF. 1 is the shortest duration and FF is the longest.

yy is the pitch of the tone. *yy* can be 0 - FF. 1 is the highest pitch and 0 or FF is the lowest.

Appendix J/ BASIC Memory Map

0000H to 25FFH	Operating System	Reserved for TRSDOS operations.
2600H to 2FFFH	Overlay Area	Used alternately by TRSDOS and BASIC. Whenever you use a TRSDOS library command, TRSDOS uses this area to store the program that will perform the command. BASIC reloads this area with its data when you return from TRSDOS.
3000H to 85FFH	BASIC	Reserved for BASIC.
8600H to Bottom of Stack	User's BASIC Program	Reserved for your programs, variables, strings, and arrays.
Bottom of Stack to HIGH\$ or User-Defined top of memory (M)	BASIC stack and File Control Block(s)	Contains the stack used by BASIC and the File Control Block(s) (FCBs). Each FCB requires 564 bytes of storage. The number of FCBs that your system has is selected with the command: BASIC (F = n), where 'n' specifies the number of files that can be open at any one time. (One additional 564-byte block is always allocated and is reserved for use by BASIC.)
User-Defined top of memory (M) or HIGH\$ to HIGH\$	Assembly language routines callable from BASIC.	This area exists only if you create it with the command, BASIC (M = address) where 'address' specifies the last address that BASIC will use. The area between "M" and HIGH\$ is used to store assembly language routines that are called by BASIC programs.

HIGH\$ to
FFFFH

Driver/Filter/User
or System tasks

Area in which drivers, filters, and tasks that are continuously used by the system are stored. Items in this area include the spooler, drivers and filters that cannot fit into the area reserved within TRSDOS, and MEMDISK (when it resides in Bank 0). Assembly language routines that are to be called from BASIC may be placed here as long as the programs follow the rules outlined in the *Technical Reference Manual*.

User Program

Your User Program space is dynamic. It is dependent on the number of data files you requested when loading BASIC (called "concurrent" files), the HIGH\$ marker, the amount of stack space, and the highest memory location you specified when loading BASIC. For information on how to load BASIC, see Chapter 1.

Assuming that the HIGH\$ marker is at the top of physical memory (FFFFH) and that the highest memory location (the 'M' option) was not specified when BASIC was loaded, then four or less concurrent files do not alter the amount of User Program space. The fifth concurrent file decreases it by 312 bytes, and six or more decrease it by 564 bytes each.

IF HIGH\$ is not at FFFF\$, or if M was specified when BASIC was loaded, then use the PRINT FRE(0) command to see the amount of User Program space available.

The number of concurrent data files also determines where the top of the stack will be. BASIC uses the following formula:

$$M - (564 \times \text{number of concurrent files}) - 564 = \text{location value}$$

The location value given by this formula is set as the top of the stack. You can set aside additional stack space by using the CLEAR statement. However, the more stack space you use, the less User Program space you will have.

Appendix K/ Using The Device-Related Commands

The advanced, device-related commands affect the assigned TRSDOS devices and the devices that you create. They are:

DEVICE, FILTER, LINK, SET, ROUTE, RESET

DEVICE is different from the other commands because instead of directly affecting the devices, DEVICE actually shows how each device is set up and what connections between devices (and files) exist. So, each time you issue one of the above commands, you should issue a DEVICE (B=ON) command to make sure the devices are set the way you want them.

Creating an Unfiltered Link

An unfiltered link is different from a filtered link because there is not an in-between program (a filter) that affects the data flowing between the two devices.

Creating an unfiltered link between a device and a file involves the ROUTE and LINK commands.

ROUTE can create a user device and routes it to a file.

LINK creates a link between two devices.

Remember that it is a good idea to issue a DEVICE command before you create a link. In the following example, we are going to route the printer. On start-up, the printer is shown in the device table as:

```
*PR => X'0DE3'
```

The device table entry shows the place in memory (X'0DE3') where the driver program that controls the printer is located. This memory address may vary.

Example

In this example we are going to link the printer to a file. That is, all data sent to the printer is also sent to the file.

To create a link between the printer (*PR) and the file PRINT/TXT:0:

1. Route the user-created device *DU to the file PRINT/TXT:0 by issuing the command:

```
ROUTE *DU TO PRINT/TXT:0 ENTER
```

The device table shows:

```
*DU <=> PRINT/TXT:0
```

The following link now exists:

```
*DU <-> PRINT/TXT
```

Everything that TRSDOS sends to *DU is sent to the file PRINT/TXT.

-
2. Link the printer to *DU, which in turn is routed to PRINT/TXT by issuing the command:

```
LINK *PR *DU ENTER
```

the device table shows:

```
*PR => *L0 ; *DU & => X'0DE3'  
*DU <=> PRINT/TXT:0
```

The following link now exists:

```
*PR -> Printer Driver (at X'0DE3')  
*PR -> *DU <-> PRINT/TXT
```

Everything that TRSDOS sends to *PR is also sent to *DU and from there to PRINT/TXT on Drive 0.

Creating a Filtered Link

Creating a filtered link involves the SET and FILTER commands. A filtered link involves a device and a filter program which affects the data that flows to or from the device.

SET prepares a user-created device for the filter connection.

FILTER creates the "logical link" between two devices. The first device is usually a system device, and the second device is always a user-created filter device.

Example

To create a filter link you need a filter program. In this example we use the system filter program KSM/FLT.

Before you issue a SET or FILTER command, be sure to issue a DEVICE command to see the start-up conditions of the system devices. In this example, we are going to filter the keyboard device. On start-up, the keyboard is shown in the device table as:

```
*KI <= X'0893'
```

The device table entry shows the place in memory (X'0893') where the driver program that controls the keyboard is located. This memory address may vary.

To create a KSM filter link between *KI and a user-created device *DU:

1. Set *DU to the KSM filter by issuing the command:

```
SET *DU KSM/FLT PRINT/DAT ENTER
```

The device table shows:

```
*KI <= X'0893'  
*DU <# [Inactive] X'FF67'  
Options: Type, KSM
```


-
2. Now use the FILTER command to connect the KSM filter program to the keyboard by issuing the command:

```
FILTER *KI *DU (ENTER)
```

The device table shows:

```
*KI <# [*DU] X'FF67'  
*DU <= X'0893'  
Options: Type, KSM
```

The following link now exists:

```
Keyboard Driver -> *DU -> KSM/FLT -> *KI
```

That is, everything that you type into the keyboard is sent through *DU, filtered through the KSM filter and then the information is available at *KI to be read by TRSDOS or a program.

3. To return the keyboard to its start-up condition, issue the command:

```
RESET *KI (ENTER)
```

4. To remove *DU from the device table, issue the following commands:

```
RESET *DU (ENTER)  
REMOVE *DU (ENTER)
```

Using the RESET Command

You can use RESET with SET, FILTER, ROUTE, or LINK. In this example, we show you what happens when you break the link between —PR and PRINT/TXT.

Example

To break the link:

```
*PR -> Printer Driver (at X'0DE3')  
*PR -> *DU <-> PRINT/TXT
```

1. First, to remove the routing between *DU and PRINT/TXT, issue the command:

```
RESET *DU (ENTER)
```

The device table shows:

```
*PR => *L0 ! *DU & => X'0DE3'  
*DU <=> NIL
```

The following link now exists:

```
*PR -> Printer Driver (at X'0DE3')  
*PR -> *DU <-> NIL
```

All output sent to *PR is still sent to *DU, even though *DU is pointed NIL.

2. To remove the link between *PR and *DU, issue the command:

```
RESET *PR ENTER
```

The device table shows:

```
*PR => X'0DE3'  
*DU <=> NIL
```

Now you have returned *PR to its original start-up condition, and the link between *PR and *DU no longer exists.

(You can type REMOVE *DU to remove *DU from the device table.)

Appendix L/ Set Up for 50 Hz AC power (non-USA users)

A utility (HERZ50) is provided for customers in areas where the AC power is 50 Hz rather than 60 Hz. It should not be used by any other customers. HERZ50 simply places a patch on the diskette that changes the clock speed for 50 Hz users.

HERZ50 is a DO-file that makes a change in the software of TRSDOS. Only the Drive 0 diskette is changed. Be sure it is write-enabled before you start the DO-file. Once the HERZ50 change is done, it will remain in effect for that diskette.

To perform the change, type:

```
DD HERZ50
```

Once the change has been made, you will need to reset the system to put the change into effect. This loads the new software into RAM.

Appendix M/ Backup Limited Diskettes

Some software products distributed by Radio Shack come on backup limited diskettes. This means that you can make only a fixed number of copies of the master diskette that you receive. You should use the master diskette to make only the backup copies that you will use, as you cannot make a backup copy of a backup that was made from a backup limited diskette.

These diskettes are clearly marked to indicate that they are backup limited. If you are uncertain, contact your Radio Shack Computer Center or the store where you purchased the diskette.

When you have exhausted the number of copies you are allowed to make or if the master diskette is write protected, the following message appears when you attempt to back up the diskette:

```
Protected source disk
```

Making a Backup Copy

Before you make a copy of a backup limited diskette, you must remove the write-protect tab from the diskette (if one is present). Because the diskette is not write-protected, you should be very careful that you do not accidentally format the master diskette or back up the blank diskette to the master diskette.

Follow the steps given below for systems with two or more floppy drives or systems with one floppy drive, as appropriate.

For systems with two or more floppy drives:

(If you have a hard disk system and two or more floppy drives, start up as a floppy disk system and use this procedure.)

1. Insert a TRSDOS system diskette into floppy Drive 0. Insert a blank diskette into floppy Drive 1.
2. Format the blank diskette, following the directions given with the FORMAT utility. (You can use the command `FORMAT :1 (Q=N) ENTER` to produce a default diskette.)

If the diskette has any flaws on it (that is, if an asterisk is displayed next to one or more cylinder numbers), repeat step 2 with another blank diskette. Remember that you can make only a fixed number of copies of this diskette, so you should try to use good media.

3. At TRSDOS Ready, type:

```
BACKUP :0 :1 (X) ENTER
```

4. When you see the prompt:

```
Insert SOURCE disk <ENTER>
```

remove the TRSDOS system diskette from Drive 0 and set it aside.

Remove the write-protect tab (if any) from the master backup limited diskette you want to copy. This will be the SOURCE diskette.

Place the backup limited diskette in Drive 0 and press **(ENTER)**.

5. The following message may appear:

```
Destination disk ID is different:
Name=diskname Date=mm/dd/yy
Are you sure you want to backup to
it <Y,N> ?
```

Respond by typing **(Y)** **(ENTER)**.

6. The computer now performs the backup. When you see the prompt:

```
Insert SYSTEM disk <ENTER>
```

remove the backup limited diskette from Drive 0 and place a write-protect tab on it.

Remove the new backup copy from Drive 1. Place a write-protect tab on it and place a label on the jacket to identify it.

Insert the TRSDOS system diskette in Drive 0 and press **(ENTER)**. A message is displayed telling you if the backup operation was successful or not. If there was an error, start over with step 1 using another blank diskette. Unsuccessful backups do not count against the number of backups you can make.

For systems with one floppy drive:

(If you have a hard disk system and one floppy drive, start up as a floppy disk system and use this procedure.)

1. Insert a TRSDOS system diskette in the drive.
2. At TRSDOS Ready, type the following command:

```
FORMAT :0 (Q=N) (ENTER)
```

3. When you see the prompt:

```
Load destination diskette <ENTER>
```

remove the TRSDOS system diskette from the drive and insert a blank diskette. Press **(ENTER)**.

4. When you see the prompt:

```
Load SYSTEM diskette <ENTER>
```

remove the formatted diskette and insert the TRSDOS system diskette. Then press **(ENTER)**.

If the disk has any flaws on it (that is, if an asterisk is displayed next to one or more cylinder numbers), repeat steps 2, 3, and 4 using another blank diskette. Remember that you can make only a fixed number of copies of this diskette, so you should try to use good media.

5. At TRSDOS Ready, type:

```
BACKUP :0 :0 (ENTER)
```

6. When you see the prompt:

```
Insert SOURCE disk <ENTER>
```

remove the TRSDOS system diskette from the drive and set it aside.

Remove the write-protect tab (if any) from the backup limited diskette you want to copy. This will be your SOURCE diskette.

Insert the backup limited diskette in the drive and press **(ENTER)**.

7. When you see the prompt:

```
Insert DESTINATION disk <ENTER>
```

remove the backup limited diskette from the drive and insert the blank diskette. Press **(ENTER)**.

8. The following message may appear:

```
Destination disk ID is different:  
Name=diskname Date=mm/dd/yy  
Are you sure you want to backup to  
it <Y,N> ?
```

Respond by typing **(Y)** **(ENTER)**.

9. You are asked to insert the SOURCE and DESTINATION disks several times. Be very careful that you do not mix them up! Simply follow the instructions in steps 6 and 7 each time you see one of the two messages.

10. When you see the prompt:

```
Insert SYSTEM disk <ENTER>
```

remove the limited backup diskette from the drive and place a write-protect tab on it. Place a write-protect tab on the new backup copy and place a label on the jacket to identify it.

Insert the TRSDOS system diskette in the drive and press **(ENTER)**. A message is displayed telling you if the backup operation was successful or not. If there was an error, start over

with step 1 using another blank diskette. Unsuccessful backups do not count against the number of backups you can make.

Backing up selected files

You can move the programs on a backup limited diskette to the hard disk using backup by class or backup reconstruct. (The latter occurs automatically when the target drive is a hard disk.) This is counted the same as making a diskette copy using the procedure described above.

Note that if you do a backup by class and move only selected files, and if any of the files that are moved are protected, it is counted as though you made a copy of the entire disk. For example, suppose that you are allowed to make three backups of a backup limited diskette. You do a backup by class to move visible files. If one of the visible files is protected, then that file is copied along with the other visible files. However, you can now make only two more copies of the files on the master disk.

For this reason, you should be careful that you do not cheat yourself out of a copy. When moving files to the hard disk from a backup limited diskette, ask for all of the files using the (SYS,INV) options in the BACKUP command. If this moves some unwanted material, it can be purged later.

You may use backup by class or backup reconstruct to move non-protected files to and from the hard disk or a backup of the backup limited diskette. However, the protected files are not backed up and will not be listed if you use the QUERY option.

SERVICE POLICY

Radio Shack's nationwide network of service facilities provides quick, convenient, and reliable repair services for all of its computer products, in most instances. Warranty service will be performed in accordance with Radio Shack's Limited Warranty. Non-warranty service will be provided at reasonable parts and labor costs.

Because of the sensitivity of computer equipment, and the problems which can result from improper servicing, the following limitations also apply to the services offered by Radio Shack:

1. If any of the warranty seals on any Radio Shack computer products are broken, Radio Shack reserves the right to refuse to service the equipment or to void any remaining warranty on the equipment.
2. If any Radio Shack computer equipment has been modified so that it is not within manufacturer's specifications, including, but not limited to, the installation of any non-Radio Shack parts, components, or replacement boards, then Radio Shack reserves the right to refuse to service the equipment, void any remaining warranty, remove and replace any non-Radio Shack part found in the equipment, and perform whatever modifications are necessary to return the equipment to original factory manufacturer's specifications.
3. The cost for the labor and parts required to return the Radio Shack computer equipment to original manufacturer's specifications will be charged to the customer in addition to the normal repair charge.

Index

- b 2-3
- ; (Advance Memory) 1-60
- (Decrement Memory) 1-60
- ! tag 2-33
- # tag 2-34
- *FR 1-35, 36, 37
 - device 1-40
 - “received Data” 1-40
- *FS 1-37, 1-44

- A —**
- A (Cancel and Restart) 2-20
- ABS 2-65
- Accessing Direct-Access File 2-56
- Action Keys 1-36
 - CLEAR 7** 1-36
 - Dump-to-disk 1-36, 44
 - *FR 1-36
 - CLEAR 8** 1-36
 - CLEAR 9** 1-37
 - CLEAR 0** 1-37
 - CLEAR :** 1-37
 - CLEAR –** 1-38
 - CLEAR SHIFT 1** 1-38
 - half-duplex 1-38
 - full-duplex 1-38
 - CLEAR SHIFT ”** 1-38
 - “Echo”ing 1-38
 - CLEAR SHIFT #** 1-39
 - carriage return 1-39
 - CLEAR SHIFT \$** 1-39
 - CLEAR SHIFT %** 1-39
 - CLEAR SHIFT &** 1-39
 - CLEAR SHIFT ’** 1-39
 - control characters 1-39
 - CLEAR SHIFT (** 1-39
 - cursor 1-39
 - CLEAR SHIFT)** 1-39
 - CLEAR SHIFT *** 1-40
 - handshaking 1-40
 - CLEAR SHIFT 0** 1-40
 - device 1-40
 - library command 1-40
 - CLEAR SHIFT =** 1-41
 - data received 1-41
- Addition 2-41
- Advanced Information ii
- Advanced Programmer’s Command 1-14
- Advanced Programmer’s Utilities 1-14
- APPEND 1-17
 - command 1-31
- Appendices A-1
- Application Keys 1-35
 - CLEAR 1** 1-35
 - CLEAR 2** 1-35
 - CLEAR 3** 1-35
 - CLEAR 4** 1-35
 - CLEAR 5** 1-35
 - CLEAR 6** 1-35
 - *FR OFF 1-37
- [arguments] 2-4
- array 2-29
- ASC 2-66
- ASCII format 1-17
- ASCII Modify 1-57
- Assigning Protection Attributes 1-20
 - to a disk 1-21
 - to a file 1-20
 - Owner passwords 1-20
 - User passwords 1-20
- ATN 2-67
- ATTRIB 1-19, 1-49
 - protection passwords 1-19
- AUTO 1-22, 2-67

- B —**
- B (Move Block of Memory) 1-57
- BACKUP 1-24
 - by class 1-25, 1-28
 - limited 1-47, A-109
 - mirror image 1-27, 1-28
 - non-limited 1-47
 - reconstruct 1-27, 1-28, 1-29
 - with the (X) parameter 1-25, 28
- BASIC 2-1
 - Command Mode 2-13
 - Concepts 2-25
 - Execution Mode 2-14
 - Functions 2-62, A-82
 - Introduction 2-59

Keywords	2-59, A-76, A-80
Line Edit Mode	2-17
Loading	2-9
Notations	2-3
Operations	2-7
reserved words	A-79
Sample Session	2-9
Terms	2-4
Variable classification	2-34
BASIC Command Mode	2-13
Special Keys	2-14
BASIC Concepts	2-25
BASIC Execution Mode	2-14
Special Keys	2-14
BASIC Line Edit Mode	2-17
Special Keys	2-18
BOOT	1-30
(BREAK) Key Handling	A-35
Breakable AUTO commands	1-30
Buffer	2-4
BUILD	1-31
Built-in drives	i
Bulletin Board Systems	1-34, 42

— C —

C (Call Instruction)	1-58
PC register	1-58
single-step	1-58
CALL	2-68
CDBL	2-69
CHAIN	2-70
Character Set	A-46
Characters per line	A-75
CHR\$	2-72
CINT	2-73
CLEAR	2-74
CLICK/FLT	A-100
CLOSE	2-75
CLS	2-76
COM/DVR	1-34, 42, A-94
COMM	1-34
Command	1-13
A (Cancel and Restart)	2-20
APPEND	1-17
Auxiliary	1-13
Break	1-46
(CONTROL) (A)	1-46

Model II, 12, or 16	1-46
(CONTROL) (C)	1-46
Model I or III	1-46
breakable AUTO	1-30
BUILD	1-31
Device Handling	1-13
device related	A-103
Diskette Handling	1-13
DO	1-31
E (Save Changes and Exit)	2-20
File Handling	1-13
I (Insert)	2-19
Initialization	1-13
Machine Language File Handling	1-13
nC (Change)	2-21
nD (Delete)	2-21
nKc (Search and "Kill")	2-22
nSc (Search)	2-22
Q (Cancel and Exit)	2-21
RESET	A-105
X (Extend Line)	2-19
COMMON	2-76
COMMunicating	
between two TRS-80's	1-44
with other computers	1-42
with mainframe	1-42
Communications	A-43
Configuration	A-85
Console	A-39
constants	2-11
CONT	2-77
CONV (CONV/CMD)	1-47
Converting to Integers	A-75
COPY	1-49
COS	2-78
CREATE	1-52
Creating	
Direct-Access Files	2-54
filtered link	A-104
Sequential-access Files	2-51
unfiltered link	A-103
CSNG	2-79
CVD, CVI, CVS	2-80
Cylinder	1-27, 1-61

— D —

D (Display)	1-58
-------------	------

D notation	2-34	Disk Files	1-7, 2-51, A-75
DATA	2-81	Disk ID's	1-27, 1-28
DATE	1-54	Disk Prompts	1-51
system	1-54	DESTINATION	1-51
today's	1-54	SOURCE	1-51
DATE\$	2-82	SYSTEM	1-51
DEBUG	1-55	Disk Read/Write Utility	1-61
activate	1-55	Division	2-40
extended	1-55	by zero	A-76
high memory	1-56	DO	1-31, 1-32, 1-72
microprocessor registers	1-56	Job Control Language	1-72
flag registers	1-56	@label	1-72
memory locations	1-57	Double Precision	2-31
PC register	1-56	:drive	A-85, 1-8
register pairs	1-56	Drivers	1-10
Debug Display	1-30	Dummy number	2-4
DEFDBL	2-83	Dummy string	2-4
DEFINT	2-83	DUMP	1-75
DEFSNG	2-83	Dump-To-Disk	1-36
DEFSTR	2-83		
DEF FN	2-84	— E —	
DEF USR	2-85	E (Save Changes and Exit)	2-20
DELETE	2-86	E notation	2-34
DEVICE	1-40, 1-65	EDIT	2-88
Delay time	1-66	Electronic Mail Services	1-34
Device section	1-65	END	2-89
Drive section	1-65	EOF	2-90
Driver	1-66	ERASE	2-91
Filter	1-66	ERL	2-92
Status section	1-65	ERR	2-93
Step rate	1-66	ERRS\$	2-93
Devices	1-10	ERROR	2-94
logical	1-10	Error Message	1-6, A-62, A-76
physical	1-10, A-85	Escape Code Sequence	1-46
devspec	1-10	EXP	2-95
*DO (Display Output (Video))	1-10	Exponentiation	2-40
*JL (Job Log)	1-10	Expressions	2-26, 2-47
*KI (Keyboard Input)	1-10	Extended Command Descriptions	1-61
*PR (Printer)	1-10	E (Enter Data)	1-61
*SI (Standard Input)	1-10	L (Locate)	1-62
*SO (Standard Output)	1-10	N (Next Load Block)	1-62
DIM	2-87	P (Print)	1-62
DIR	1-28, 1-68	T (Type ASCII)	1-63
<i>partspec</i>	1-68	V (Compare)	1-63
Direct-Access Files	2-54	W (Word)	1-63
Accessing	2-56	/extension	1-8
Creating	2-54		

— F —

F (Fill Memory)	1-58
FIELD	2-96
file-for-file copy	1-28
File Handling Commands	1-13
<i>filename</i>	1-8
Files	
BASIC ASCII	1-47
combining	A-23
data	1-47
fragmented	1-28
Filespec.	1-7
FILTER	1-77
<i>phantom devspec</i>	1-77
Filters	1-10
FIX	2-97
FLOPPY/DCT.	A-99
Hard disk installations	A-99
Floppy Disk Drive	A-40
Floppy Drive 0	1-30
FOR/NEXT	A-76, 2-98
FORMAT.	1-24, 1-31, 1-78
erase all data from disk	1-78
prepare a new disk	1-78
Format Prompts	1-78
FORMS	1-82
FORMS/FLT.	A-95
FRE	2-100
FREE	1-28, 1-85
Functions	2-27, 47, 62
control	A-45
graphics	A-45
keys	1-35
action	1-36
applications	1-35
text	A-45

— G —

G (Go to an Address/Execute)	1-58
GET	2-100
GOSUB	2-101
GOTO	2-102
Graphics Characters	1-42

— H —

H (Hex Modify)	1-58
hexadecimal value	1-59

vertical bars	1-58
HANDSHAKE	1-37, 40, 44
HERZ50	A-107
Hex byte representations	1-32
HEX\$	2-103
Hexadecimal value	1-34

— I —

I (Insert)	2-19
I (Single-Step Execution)	1-59
If highlighted	1-15
IF . . . THEN . . . ELSE	A-77, 2-103
Initialization commands	1-13
INKEY\$	2-105
INP	2-105
INPUT	2-106
INPUT#	2-107
INPUT\$	2-108
INSTR	2-110
INT	2-111
Integer	2-4
Integers	2-30

— J —

J (Jump)	1-59
JCL Compiling	A-12
Advanced	A-25
description and terms	A-13
compile phase	A-13
execution	A-13
SYSTEM/JCL	A-12
label	A-13
logical operator	A-14
token	A-13, A-16
using logical operators	A-26
Job Control Language	A-3
creating	A-4
restrictions	A-4
simple execution	A-4
using labels	A-24
JOBLOG	A-91

— K —

Keyboard	A-40
KILL	2-112
KSM file	1-32
KSM/FLT	1-31, A-92

building a file	A-92	//EXIT	A-7
— L —		//FLASH	A-7
LEFT\$	2-112	//INPUT	A-7
LEN	2-113	//KEYIN	A-7
LET	2-114	//number	A-11
LIB	1-87	//PAUSE	A-8
Technical Information	1-87	//SLEEP	A-8
Line	2-4	//STOP	A-8
logical	1-32	/// (triple slash)	A-11
physical	1-32	//WAIT	A-9
LINE INPUT	2-115	keyboard	A-5
LINE INPUT#	2-116	nested //IF	A-27
LINK	1-88	//INCLUDE	A-28
device to a file	1-88	pause/delay	A-5
LIST	1-90, 2-117	Main memory usage	1-45
LLIST	2-118	FIFO storage compartment	1-45
LOAD	1-92, 2-119	HIGH\$	1-45
Load module format	1-17	Marker, end of file	1-31
Loading BASIC	2-9	MEM	2-125
Loading the Program	2-12	MEMDISK/DCT	A-96
LOC	2-120	disable	A-98
LOF	2-121	double density	A-97
LOG	2-122	installing	A-97
LOC/CMD	1-93	single density	A-97
hard disk installations	1-93	Technical Information	A-99
Logical device	1-10	MEMORY	1-94
Logical line	1-32	HIGH\$	1-94
LPOS	2-123	LOW\$	1-94
LPRINT, LPRINT USING	2-123	Memory Map	A-101
LRL	1-49, 1-52	MENU	1-36
LSET	2-124	MERGE	2-126
— M —		MID\$... (Function)	2-129
MACROS	A-5	MID\$... (Statement)	2-128
alert	A-6	Mirror Image Backup	1-27, 1-28
comment	A-5	MKD\$, MKI\$, MKS\$	2-130
conditional	A-15	MOD flags	1-27
comment	A-15	Modem	1-42
higher order logical	A-15	Multiplication	2-40
logical	A-15	— N —	
merge	A-15	NAME	2-131
termination	A-15	nC (Change)	2-21
execution	A-5	nD (Delete)	2-21
//ABORT	A-6	Nested subroutines	A-76
//ALERT	A-6	NEW	2-131
//DELAY	A-6	News and Information System	1-34
		nKc (Search and "KILL")	2-22

Non-ending Loops..... 1-50
 Notations (BASIC)..... 2-3
 nSc (Search) 2-22
 Number 2-4
 Numeric Operators 2-39, 2-42

— O —

O (Return to TRSDOS Ready)..... 1-59
 OCT\$..... **2-132**
 ON ERROR GOTO..... **2-132**
 ON ... GOSUB **2-133**
 ON ... GOTO **2-134**
 O'NNNN 2-3
 OPEN **2-135**
 Operating Temperature A-41
 Operators 2-39
 Logical..... 2-44
 Numeric..... 2-39, 2-42
 Relational 2-42
 String..... 2-42
 OPTION BASE **2-136**
 Options for Loading BASIC..... 2-9
 OUT..... **2-137**
 Owner passwords 1-19

— P —

Parameters..... 1-15
 [parameters]..... 2-4
 Parentheses..... 2-45
 Partspec 1-8, 1-28
 "wildcard" mask (\$) 1-8
partspec 1-24, 1-47
 .password 1-8
 Passwords 1-20
 Owner 1-20
 User 1-20
 PATCH **1-96**
 direct modify..... 1-99
 memory load location..... 1-97
 PATCH utility 1-31
 Pause Transmission..... 1-40
 PEEK..... **2-137**
 Peripheral Interfaces A-41
 Physical device 1-10
 Physical line..... 1-32
 POKE..... **2-138**
 Ports A-76

POS..... **2-138**
 Power Supply..... A-40
 Power-up configuration 1-30
 PRINT **2-139**
 PRINT TAB..... A-77, **2-145**
 PRINT USING **2-141**
 Print Zones..... A-75
 PRINT @ A-77, **2-144**
 PRINT#..... **2-146**
 Printing
 double-precision numbers..... A-76
 single-precision numbers..... A-76
 Protection passwords 1-19
 PURGE **1-100**
 PUT **2-147**

— Q —

Q (Cancel and Exit) 2-21
 Q (Port)..... 1-59
 Quick reference card 1-36
 Quick reference label 1-41

— R —

R (Register Pair) 1-60
 register pair codes 1-60
 RANDOM **2-148**
 READ..... **2-149**
 Receiving large files
 from another computer..... 1-46
 Relational Operators..... 2-42
 REM..... **2-150**
 REMOVE **1-102**
 user-created device 1-102
 RENAME..... **1-103**
 RENUM..... **2-151**
 REPAIR (REPAIR/CMD) **1-105**
 Reserved Words A-76, 2-29
 RESET 1-18, **1-106**
 improperly closed files 1-106
 RESTORE **2-152**
 RESUME..... **2-153**
 Resume transmission..... 1-40
 RETURN..... 2-154
 Reverse video A-46
 RIGHT\$..... 2-154
 RND 2-155
 ROM Subroutines A-75

ROUTE	1-108	Substitution Fields.....	A-21
ROW	2-156	Subtraction	2-41
RS-232C.....	i, 1-113	SWAP	2-165
communications line.....	1-34	Symbol	
modem	1-113	DC1	1-40
serial printer	1-113	resume transmission	1-40
Technical Information.....	1-114	DC2	1-40
RSET.....	2-157	*FR device ON	1-40
RUN.....	1-110, 2-157	DC3	1-40
		pause transmission.....	1-40
— S —		DC4	1-40
S (Full Screen Mode).....	1-60	*FR device OFF	1-40
SAVE.....	2-158	using % symbol.....	A-29
Saving the Program	2-11	Syntax	1-15, 2-4
Screen Print	1-7	SYSGEN.....	1-120
Sector	1-61	configuration file	1-120
Sequential-Access Files.....	2-51	Sysgened configuration	1-29
Creating.....	2-51	SYSTEM.....	1-122, 2-166
Updating	2-53	Alive.....	1-122
SET	1-111	Blink.....	1-122
driver program	1-111	Date.....	1-123
filter program	1-111	Drive	1-123
SETCOM.....	1-42, 1-113	SYSRES	1-124
SETKI	1-116	SYSTEM.....	1-125
SGN.....	2-159	TIME	1-125
Simple Variables	2-29	TRACE	1-125
SIN.....	2-160	TYPE[=switch].....	1-125
SOUND.....	2-160	(SYSRES=number).....	1-28
SPACE\$	2-161	System modules	1-28
SPC	2-162		
Special Characters	A-58	— T —	
SPOOL	1-117	TAB	2-167
disk buffer.....	1-117	TAN	2-167
memory buffer	1-117	TAPE100.....	1-126
output buffer.....	1-117	Model 100 Computer	1-126
SQR.....	2-162	Technical Information.....	ii
Statements	2-26, 2-60	TIME	1-127
Static Electricity	A-62	clock.....	1-127
STOP.....	2-163	TIMES\$	2-168
String	2-4	Timesharing Systems.....	1-34
String Operator	2-42	TROFF, TRON.....	2-169
String Relations	2-43	TRSDOS.....	1-1
String Space.....	A-76	Abbreviations	1-6
Strings	2-32	application programs	1-6
STR\$	2-164	date	1-6
STRING\$	2-164	Notations.....	1-5
Subscripted Variables	2-29	Terms	1-5

<i>command</i>	1-5	Utilities.....	1-14
<i>devspec</i>	1-5		
<i>disk</i>	1-5	— V —	
<i>disk.ID</i>	1-6, 1-28	VAL	1-173
<i>diskette</i>	1-5	Variable Names.....	A-75, 2-29
<i>filespec</i>	1-5	<i>Variables</i>	2-28
I/O	1-6	VARPTR.....	2-174
(<i>parameters</i>)	1-5	VERIFY.....	1-128
Type Declaration Tags.....	2-33, 2-35	Video Display.....	A-39
! tag	2-33		
# tag	2-34	— W —	
D notation.....	2-34	WAIT	2-177
E notation	2-34	WHILE ... WEND.....	2-178
		Wildcard mask (\$).....	1-8
— U —		WRITE.....	2-179
U (Update)	1-60	WRITE#	2-180
Updating Sequential-Access Files.....	2-53	Write-enabled.....	1-23
User passwords.....	1-20		
USING		— X —	
//ASSIGN	A-15	X (Extend Line)	2-19
//. Comment and //QUIT	A-19	X (Return).....	1-60
//IF, //END, //ELSE	A-15	X'NNNN.....	2-3
//SET and // RESET	A-15		
USR.....	2-170		