# About MS-DOS Version 02.11.03

This new version of the Tandy 2000 MS-DOS Operating System contains new versions of some programs for enhancements and to correct some previous problems. In addition, some programs have been replaced with programs that are more compatible with other Tandy MS-DOS computers.

Documentation on the changes and new features is provided in a file, README.DOC, on the MS-DOS disk. To view the contents of the file on the screen, type at the system prompt:

```
TYPE README.DOC | MORE  < ENTER >
```

However, because this file contains a lot of useful information, you'll probably want to print it out. If you have a printer attached to your computer, type at the system prompt:

```
PRINT README.DOC  < ENTER >
```

**Thank you**
**Tandy Corporation**
**10/86**

# Addendum to the Tandy 2000 BASIC Reference Manual

Cat. No. 26-5103

The Tandy 2000 BASIC interpreter software provided in this package includes many enhancements to the previous version. This addendum documents BASIC's new statements and functions. Please add this new information to your current reference manual.

ENVIRON "parameter id=text" [;"parameter id=text" ,...

Lets you modify BASIC's Environment String Table, such as changing the PATH parameter for a child process or passing parameters to a child process. BASIC's Environment String Table is initially empty.

Parameter id is the name of the parameter.

Text is the new parameter text. It must be separated from parameter id by an equal sign (=) or a space. BASIC reads the first nonblank, nonequal sign character after the parameter id as the text. If you omit text, or specify a null string or a semicolon (;), BASIC removes the parameter from the Environment String Table and compresses the table.

Parameter id = text must be enclosed in quotation marks and be typed in all uppercase characters.

When you change a parameter in the Environment String Table, BASIC deletes the old parameter and adds the new one to the end of the table.

If the parameter does not exist in the Environment String Table, BASIC adds it to the end of the table.

For more information on Environment String Tables, see the Programmer's Reference manual for your computer (Cat. No. 26-5403). It is available at your Radio Shack Computer Center.

**Examples**

```
ENVIRON "PATH=A:\"
```

sets the default path to the root directory on Drive A.

```
ENVIRON "SALES=MYSALES"
```

sets the name SALES equal to MYSALES. The Environment String Table now looks like this:

```
PATH=A:\;SALES=MYSALES
```

**ENVIRON$** [("parameter id")] [(number)]

Returns the specified environment string from BASIC's Environment String Table.

Parameter id specifies the parameter for which to search. ENVIRON$ returns the text string for parameter id. If the parameter does not exist or does not contain a text string, ENVIRON$ returns an empty string. Parameter id must be enclosed in quotation marks. If you omit parameter id, you must specify number.

Number specifies which parameter to return by its position within the table. ENVIRON$ returns the text string for the number parameter. If there is not a parameter in that position, ENVIRON$ returns an empty string. If you omit number, you must specify parameter id.

Parameter id and number are mutually exclusive, only one may be specified on the command line.

For more information on Environment String Tables, see the Programmer's Reference manual for your computer (Cat. No. 26-5403). It is available at your Radio Shack Computer Center.

**Example**

If you execute the following ENVIRON statements:

```
ENVIRON "PATH=A:\"
ENVIRON "SALES=MYSALES"
```

the Environment String Table looks like this:

```
PATH=A:\;SALES=MYSALES
```

The command PRINT ENVIRON$("PATH") prints A:\.

The command PRINT ENVIRON$(2) prints SALES=MYSALES.

## ERDEV

Returns the value of a device error within MS-DOS as set by the Interrupt 24 handler. The lower 8 bits of ERDEV contain the Interrupt 24 error code.

For more information on device drivers and errors, see the Programmer's Reference manual for your computer (Cat. No. 26-5403). It is available at your Radio Shack Computer Center.

See also ERDEV$.

## ERDEV$

Returns the name of the device (as set by the Interrupt 24 handler) when a device error occurs.

If the error occurred on a character device, ERDEV$ returns the 8-byte character device name.

If the error does not occur on a character device, ERDEV$ returns the 2-character block device name.

For more information on device drivers and errors, see the Programmer's Reference manual for your computer (Cat. No. 26-5403). It is available at your Radio Shack Computer Center.

See also ERDEV.

IOCTL [#]buffer,string

Sends a control data string to a device driver. Control data can be sent to a drive only after it has been opened.

Buffer is the number assigned to the driver when you opened it. The number sign (#) is optional. It is provided for compatibility with other BASICs.

String is a string expression containing a series of commands called "control data." The commands are generally 2 to 3 characters long and may be followed by an alphanumeric argument. The commands are separated by semicolons (;). String may be a maximum of 255 bytes.

For more information on device drivers, see the Programmer's Reference manual for your computer (Cat. No. 26-5403). It is available at your Radio Shack Computer Center.

**Example**

If you write your own driver to replace PRN to set the page length, the IOCTL command may be:

PLn where n is the new page length.

To open the new PRN driver and set the page length at 56 lines per page, use the following statements:

```
10 OPEN "PRN" FOR OUTPUT as 1
20 IOCTL 1,"PL56"
```

IOCTL$([#]buffer)

Returns the control data string from a device that you have opened previously.

Buffer is the number assigned to the driver when you opened it. The number sign (#) is optional. It is provided for compatibility with other BASICs.

You can use the IOCTL$ function to confirm that a IOCTL statement succeeded (or failed). You can also use IOCTL$ to get information from the device.

For more information on device drivers, see the Programmer's Reference manual for your computer (Cat. No. 26-5403). It is available at your Radio Shack Computer Center.

**Example**

```
10 OPEN "\DEV\PRN" AS 1
20 IF IOCTL$(1) = "NR" THEN PRINT "PRINTER NOT READY"
```

## LCOPY

Copy all text data on the screen to the printer.

### Sample Program

```
550 FOR I=1 TO 24
560 PRINT STRING$(79,33)
570 NEXT I
580 LCOPY
```

This program segment prints exclamation points on the screen, and dumps them to the printer.

MKDIR dirpath

Creates the directory specified by dirpath.

Dirpath is a standard directory specification as described in Chapter 1. If you omit the drive indentifier, the directory is created on the current drive. If you omit the root directory symbol (\), the directory is created in the current directory.

**Examples**

```
MKDIR "A:\ACCTS\PAYABLE"
```

creates the directory PAYABLE in the ACCTS directory on Drive A.

```
MKDIR "\ADDRESS"
```

creates the directory ADDRESS in the root directory on the current drive.

```
MKDIR "NAMES"
```

creates the directory NAMES in the current directory on the current drive.

**ON PLAY(number) GOSUB line**

Transfers program control to a subroutine when the number of notes in the background music buffer goes from number to number minus 1. This event trapping allows continuous music by letting you maintain a full music buffer.

Number is an integer in the range 1 to 32, indicating that control should transfer to line when the number of notes left in the music buffer is less than number.

Line is the first line of the subroutine to execute when the number of notes in the music buffer is less than number. If you specify Line 0, you turn off play event trapping. Use the RETURN statement to exit the subroutine.

BASIC executes the ON PLAY() GOSUB statement only when playing background music (PLAY "MB") and if the PLAY ON statement has been executed to enable event trapping.

If a PLAY STOP statement has been issued to halt event trapping temporarily, BASIC executes the subroutine immediately after the next PLAY ON statement.

When you execute the ON PLAY() GOSUB statement, BASIC immediately issues a PLAY() STOP to prevent recursive traps. When BASIC executes the RETURN from the subroutine, it automatically executes another PLAY() ON statement to enable trapping again, unless the subroutine executes a PLAY() OFF statement.

> **Notes:** BASIC does not issue a play event trap if the background music queue is already empty when you execute a PLAY ON.
>
> The PLAY statement is supported by a 32-element music queue. Given that "normal" and "staccato" notes are constructed from 2-note elements, the queue can contain as few as 16 notes or as many as 32 notes.

Therefore, select conservative values for the trap number. For example, if number is set at 32, event traps might happen so often that there is little time to execute the rest of your program. It is suggested that the trap number be less than 16 for better performance.

**Example**

```
10 PLAY ON
20 ON PLAY(2) GOSUB 1000
30 REM
.
.
.
500 END
1000 REM PROCESSING ROUTINE
.
.
.
1100 RETURN 30
```

Line 10 turns on play trapping. After each program statement is executed, BASIC checks to see if the number of notes in the music buffer is less than 2 notes. If it is, BASIC immediately executes the subroutine at Line 1000.

**ON TIMER** (number) **GOSUB** line

Transfers program control to a subroutine when the specified period of time has elapsed.

Number indicates the number of seconds. Number may be a value in the range 1 to 86400 (86400 seconds = 24 hours).

Line is the first line number in the subroutine to execute when the specified time has passed. If you specify Line 0, you turn off trapping for the timer. Use RETURN to exit the subroutine.

BASIC executes the ON TIMER() GOSUB statement only if a TIMER ON statement has been executed previously to enable time event trapping.

If a TIMER STOP statement has been issued to halt time event trapping temporarily, BASIC executes the subroutine immediately after the next TIMER ON statement.

When you execute the ON TIMER() GOSUB statement, BASIC immediately issues a TIMER STOP to prevent recursive traps. When BASIC executes the RETURN from the subroutine, it automatically executes another TIMER ON statement to enable trapping again, unless the subroutine executes a TIMER OFF statement.

**Example**

```
10 TIMER ON
20 ON TIMER (60) GOSUB 1000
30 REM
.
.
.
500 END
.
.
.
1000 REM PROCESSING ROUTINE
.
.
.
1100 RETURN 30
```

Line 10 turns on timer trapping. After each statement is executed, BASIC checks to see if the specified time has elapsed. If it has, BASIC immediately executes the subroutine at Line 1000.

PLAY action

Turns on, turns off, or temporarily halts background music event trapping.

Action may be any of the following:

ON enables play event trapping.

OFF disables play event trapping.

STOP temporarily suspends play event trapping.

Use the PLAY/Trap statement in a background music trap routine with the ON PLAY GOSUB statement to detect when the number of notes in the background music queue goes from number to number minus 1.

The PLAY ON statement turns on the trap. BASIC checks the number of notes in the background music queue after each program line. If the number is equal to that in the ON PLAY() GOSUB statement, BASIC transfers program control to the line number specified.

The PLAY STOP statement temporarily halts background music trapping. If the number of notes equals the specified number, BASIC does not transfer program control to the ON PLAY() GOSUB statement until you turn on trapping again by executing a PLAY ON statement. BASIC remembers that the number of notes was equal and branches to the subroutine immediately after trapping is turned on again

The PLAY OFF statement turns off background music trapping. BASIC does not remember if the number of notes in the queue is equal to the number specified when trapping is turned on again.

See ON PLAY() GOSUB for more information about background music trapping.

**PMAP** **Function**

**PMAP**(coordinate,action)

Returns the physical or world coordinate for the specified coordinate.

Coordinate is any x- or y-coordinate. If coordinate is a physical coordinate, it must be within the limits of the screen. If coordinate is a world coordinate, it may be any single precision floating point number.

Action is one of the following:

0 returns the physical x-coordinate for the specified world coordinate.

1 returns the physical y-coordinate for the specified world coordinate.

2 returns the world x-coordinate for the specified physical coordinate.

3 returns the world y-coordinate for the specified physical coordinate.

**Example**

```
A = PMAP(200,0)
```

returns the physical x-coordinate of the world coordinate 200 and places it in A.

**RMDIR** dirpath

Removes (deletes) the directory specified by dirpath.

Dirpath is a standard directory specification as described in Chapter 1. If you omit the drive identifier, the directory is deleted from the current drive. If you omit the root directory symbol (\), the directory is deleted from the current directory.

The directory being deleted must be empty except for the "." and ".." symbols. Use the MS-DOS COPY command to move those files you want to save; then use KILL to remove all files from the directory.

**Examples**

```
RMDIR "A:\ACCTS\PAYABLE"
```

removes the directory PAYABLE from the ACCTS directory on Drive A.

```
RMDIR "\ADDRESS"
```

removes the directory ADDRESS from the root directory on the current drive.

```
RMDIR "NAMES"
```

removes the directory NAMES from the current directory on the current drive.

SHELL [command]

Loads and executes another program (.EXE or .COM) or an internal command as a child process to the original program. After the child process ends, control returns to the BASIC program at the statement following the SHELL statement.

Command is a string expression containing the name of the program you want to run. You may also specify command arguments on the command line. Use a space to separate arguments from the program name. If you omit command, SHELL transfers control to COMMAND. You can now execute MS-DOS commands as allowed by COMMAND. To return to BASIC, use the MS-DOS EXIT command.

SHELL sends the command information to COMMAND.COM the MS-DOS command processor. IF you omit the extension in the program name, COMMAND looks for the program with a .COM extension, then with an .EXE extension and finally with a .BAT extension. If COMMAND still cannot find the program, it issues a "File not found" error to SHELL.

> **Note:** Do not specify BASIC as the command string of SHELL. If you do, BASIC might not function properly.

For more information on child processes and COMMAND.COM, see the MS-DOS Reference and the Programmers's Reference manuals for computer (Cat. No. 26-5403). They are available through your Radio Shack Computer Center.

**Examples**

```
SHELL
```

transfers control to COMMAND.COM You can execute MS-DOS commands such as:

```
DIR
TIME
```

and then type EXIT to return to BASIC.

The following command uses redirection of input and output and the MS-DOS SORT command.

```
SHELL"SORT <data.in>data.out"
```

sorts the text from data.in and writes it to data.out.

**TIMER**

Returns the number of seconds since midnight or since the last system reset.

BASIC always returns a single precision number.

you can use TIMER as the argument for the RANDOMIZE statement to reseed the random number generator. See RANDOMIZE for more information.

**Example**

```
A = TIMER
```

stores the number returned by TIMER into variable A.

**TIMER** action

Turns on, turns off, or temporarily halts timer event trapping.

Action may be any of the following:

| | |
|---|---|
| ON | enables timer event trapping. |
| OFF | disables timer event trapping. |
| STOP | temporarily suspends timer event trapping. |

The TIMER ON statement turns on the trap. BASIC checks the value of timer after each program line. If the number is equal to that in the ON TIMER( ) GOSUB statement, BASIC transfers program control to the line number specified.

The TIMER STOP statement temporarily halts timer trapping. If the timer equals the specified number, BASIC does not transfer program control to the ON TIMER( ) GOSUB statement until you turn on trapping again by executing a TIMER ON statement. BASIC remembers that the timer value was equal and branches to the subroutine immediately after trapping is turned on again.

The TIMER OFF statement turns off timer trapping. BASIC does not remember if the value of timer equals the number specified when trapping is turned on again.

**Sample Program**

See ON TIMER( ) GOSUB for an example.

**VIEW** [SCREEN] (x1,y1)-(x2,y2)[,[color][,[border]]]

Creates a viewport that redefines the screen parameters. This defined area, a window, becomes the only place you can draw graphic displays.

(x1,y1) specifies the upper-left coordinates for the rectangular viewport.

(x2,y2) specifies the lower-right coordinates for the rectangular viewport.

All coordinates must be within the limitations of the screen.

Color lets you fill in the specified viewport with specified color.

Border is an integer in the range 0 to 15.

SCREEN specifies that all coordinates used in drawing are absolute to Point 0,0 on the screen. If you omit SCREEN, all coordinates specified are relative to the viewpoint coordinates.

If you omit all options, BASIC sets the viewport to define the entire screen.

**Examples**

```
VIEW (10,10)-(100,100)
```

sets up a viewport with the upper-left corner at 10,10 and the lower-right corner at 100,100. Since SCREEN is omitted, all subsequent coordinates are relative to the viewport. For example, PSET (5,5),3 actually sets point 15,15.

```
VIEW SCREEN (20,25)-(100,150)
```

sets up a viewport. Because SCREEN is specified, all subsequent coordinates are absolute. For example, PSET (5,5),3 does not appear because it is outside the viewport. PSET (30,30),3 is within the viewport.

**Notes:**

- BASIC ignores any points that are outside the viewport's limits.
- RUN, SCREEN, and WINDOW statements, without parameters, define the entire screen as the viewport.
- CLS clears only the active viewport.

**Sample Program**

```
10 SCREEN 1
20 VIEW (10,10)-(200,100),2
30 PSET (1000,50)
40 DRAW "L40 E20 F20"
```

Line 20 sets the viewport. Line 30 sets the starting point for the DRAW statement in Line 40.

**VIEW PRINT** top line **TO** bottom line

Creates a text viewport that redefines the text screen parameters. All statements and functions that normally function within the text viewport now function in the new text screen parameters. Cursor movement and scrolling are also limited to the text viewport.

Top line specifies the first line of the text viewport. It may be in the range 1 to 24, but must be less than bottom line. If you omit top line, BASIC assumes Line 1 as the beginning of the text viewport.

Bottom line specifies the last line of the text viewport. It may be in the range 1 to 24, but must be greater than the top line. If you omit bottom line, BASIC assumes Line 24 as the end of the text viewport.

If you omit all parameters, VIEW PRINT defines the entire screen as the text viewport.

**Example**

```
VIEW PRINT 1 TO 15
```

BASIC defines the first 15 lines of the video as the text viewport. All cursor movements, scrolling, and text screen functions and statements are limited to these boundaries.

**WINDOW** [SCREEN] [(x1,y1)-(x2,y2)]

Lets you change the physical coordinates of the screen (or current viewport) by defining "world coordinates." World coordinates can be any single-precision floating point numbers, including numbers outside the physical range of the screen as defined by the VIEW statement.

> **Note:** The viewport is set to the entire screen by default. For more information on viewports, see the VIEW command

(x1,y1) specifies the world coordinates for the upper-left corner of the screen. x is the horizontal coordinate, and y is the vertical coordinate.

(x2,y2) specifies the world coordinates for the lower-left corner of the display. x is the horizontal coordinate, and y is the vertical coordinate.

The SCREEN option tells BASIC to set the coordinates like the screen display where the lesser y-coordiante is in the upper-left corner of the screen. If you omit screen, BASIC inverts the y-coordinates to show a true cartesian coordinate system. That is, the lesser y-coordinate is in the lower-left corner of the screen.

WINDOW lets you plot points outside the normal screen coordinate limits by setting new world coordinates to the screen. WINDOW transforms the new world coordinates onto the screen, usually altering the aspect ratio.

> **Note:** CIRCLE, GET, and PUT do not use world coordinates.

You can easily plot graphs by specifying coordinates that are directly proportional to the limits of the graph. For example, to plot the increase of sales from 1984 to 1987 with sales averaging 100,000 to 300,000, you can use the following command:

```
WINDOW (1984,100000)-(1987,30000)
```

The coordinates can be pictured for commands that use world coordinates:

1984,30000

1984,100000 _____ 1987,100000

If you give the command:

**WINDOW SCREEN (1984,100000)-(1987,300000)**

the coordinates can be pictured as follows for commands that use world coordinates:

1984,100000

1984,300000 _____ 1987,300000

**Note:** RUN, SCREEN, and WINDOW statements, without parameters, define the entire screen as the window.