THE IORESULT FUNCTION

This function returns an integer value which reflects the status of the last completed I/O operation. The form is

```
IORESULT
```

The values returned by IORESULT are as follows (also see Table 2):

Ø	No error; normal I/O completion
1	Bad block on diskette (not used on Apple)
2	Bad device (volume) number
3	Illegal operation (e.g., read from PRINTER:)
4	Unknown hardware error (not used on Apple)
5	Lost device -- no longer on line
6	Lost file -- file is no longer in directory
7	Bad title -- illegal filename
8	No room -- insufficient space on diskette
9	No device -- volume is not on line
1Ø	No such file on specified volume
11	Duplicate file title
12	Attempt to open an already open file
13	Attempt to access a closed file
14	Bad input format -- error in reading real or integer
15	Ring buffer overflow -- input arriving too fast
16	Write-protect error -- diskette is write-protected
64	Device error -- bad address or data on diskette

In normal operation, the Compiler will generate code to perform run-time checks after each I/O operation except UNITREAD, UNITWRITE, BLOCKREAD, or BLOCKWRITE. This causes the program to get a run-time error on a bad I/O operation. Therefore if you want to check IORESULT with your own code in the program, you must disable this compiler feature by using the (*$I-*) option (see Chapter 4).

Note that IORESULT only gives a valid return the first time it is referenced after an I/O operation. If it is referenced again (without another I/O operation), it will always return Ø.

INTRODUCTION TO TEXT I/O

In addition to PUT and GET, Apple Pascal provides the standard procedures READ, READLN, WRITE, and WRITELN, collectively known as the text I/O procedures. They perform the same tasks as in standard Pascal and have the same syntax (with the addition of STRING variables). However, the details of their operation are specific to Apple Pascal and can be complicated. Also, the use of STRING variables and the distinction between TEXT and INTERACTIVE files have important effects.

The text I/O procedures can only be used with files of type TEXT or INTERACTIVE. As already mentioned, RESET makes a distinction between these two file types: when a TEXT file is RESET, a GET is automatically performed but when an INTERACTIVE file is RESET, no GET is performed. This requires READ and READLN to be rather complex procedures. Like many other complex creatures, they will behave simply if you use them simply. Therefore, this manual is written with some assumptions in mind about how they will be used. These assumptions can be translated into the following specific suggestions:

- When using the text I/O procedures don't use GET or PUT, and don't refer explicitly to the file buffer variable F^. The reason is that the text I/O procedures themselves use GET and PUT in complicated ways.

- Don't mix reading and writing operations on the same diskette textfile. If you read from a textfile, CLOSE it and reopen it before writing to it; and vice versa.

- To open an existing diskette textfile for reading, always use RESET. To open an existing diskette textfile for writing, always use REWRITE.

- Don't use READ with a STRING variable. Use READLN.

- Don't use the EOLN function with READLN, and don't use it with STRING variables.

If you follow these suggestions, the text I/O procedures will work exactly as described in the following pages. These are not rules of Pascal; there is nothing in the system that will enforce them. However, the exact details of what happens if you ignore the suggestions are beyond the scope of this chapter.

There may be situations in which these assumptions and suggestions are too restrictive. If so, you will need the complete details on how READ and READLN behave in all possible situations, as given in Appendix C.

In particular, you need the information in Appendix C if you want to mix reading and writing operations or overwrite part of an existing text file without destroying all of the original contents.

THE READ PROCEDURE

This procedure may be used only on TEXT (FILE OF CHAR) or INTERACTIVE files. It allows characters and numeric values to be read from a file without the need for explicit use of GET or explicit reference to the window variable. The form is

```
PROCEDURE READ ( [ FILEID, ] VBLs )
```

where FILEID is the identifier of a TEXT or INTERACTIVE file which must