Defining Your Function Keys

The key to productivity

Anybody who has worked with PCs has doubtlessly gotten to know the special function keys on both the standard and enhanced keyboards. Various software packages have saved the user countless keystrokes by programming the keys to perform a multitude of otherwise mundane activities.

The bad news is that DOS doesn't really take advantage of the function keys at the operating system level. Aside from recalling characters from the command buffer with the F1 and F3 keys, the rest of the keys are really wasted. If you spend as much time on your PC as we do, you probably have quite a few ideas for customizing these keys. By using the techniques presented here, you'll be able to program the function keys as well as the other special keys (i.e., Home, End, Page Up, Page Down, etc.).

REDEFINING THE KEYS

Before you begin, you must make sure that the ANSI.SYS device driver is loaded each time you turn on your PC. There should be a line in your CONFIG.SYS file that reads something like this:

```
device = ansi.sys
```

If you don't have this line in your CONFIG.SYS file, then you'll have to add it. If you don't even have a CONFIG.SYS file, you must create one. The easiest way to do this is to type

```
COPY CON CONFIG.SYS
device = ansi.sys
```

and then press Control-*Z* (hold down the Control key and press Z). What you typed at the keyboard will then be saved in the CONFIG.SYS file. If the ANSI.SYS driver is located in a subdirectory on your hard disk, you must specify the directory. For example,

```
device = \dos\ansi.sys
```

by Antonio and Alfonso De Pasquale
223 Bennington St., Apt #1, East Boston, MA 02128.

Once you have made sure that ANSI.SYS is loaded, you're ready to program your function keys.

The basic trick to programming these keys is to use the DOS PROMPT statement in conjunction with the Escape character and the key's sequence code. The syntax for the command is as follows:

```
prompt $e[key sequence code;"command to execute$_"p
```

For example, the following statement will redefine the F1 key to execute a DIR statement.

```
prompt $e[0;59;"DIR$_"p
```

In this statement, PROMPT is the DOS command for changing the system prompt; $e denotes the escape sequence; the left bracket separates the function key code from the escape sequence; 0;59 is the sequence code for the F1 key; the second semicolon (;) separates the sequence code from the command to be executed; "DIR$_" is the DOS directory command; and *p* completes the prompt (note that

Table 1: Key Sequence Codes

Key	Alone	Shift	Control	Alt
F1	0;59	0;84	0;94	0;104
F2	0;60	0;85	0;95	0;105
F3	0;61	0;86	0;96	0;106
F4	0;62	0;87	0;97	0;107
F5	0;63	0;88	0;98	0;108
F6	0;64	0;89	0;99	0;109
F7	0;65	0;90	0;100	0;110
F8	0;66	0;91	0;101	0;111
F9	0;67	0;92	0;102	0;112
F10	0;68	0;93	0;103	0;113
F11	0;133	0;135	0;137	0;139
F12	0;134	0;136	0;138	0;140
Home	0;71	55	0;119	n/a
Up Arrow	0;72	56	n/a	n/a
Page Up	0;73	57	0;132	n/a
Left Arrow	0;75	52	0;115	n/a
Right Arrow	0;77	53	0;114	n/a
Down Arrow	0;80	54	0;116	n/a
End	0;79	49	0;117	n/a
Page Down	0;81	51	0;118	n/a
Insert	0;82	48	n/a	n/a
Delete	0;83	46	n/a	n/a

this last letter must be a lower-case *p*). In the directory command, the dollar sign and underscore ($_) instruct DOS to perform a carriage return. Notice that the command was placed inside quotation marks.

If you typed this command correctly, you can now display a directory by simply hitting the F1 key. However, you may have also noticed that your original DOS prompt has disappeared. You can recall it, if you like, by issuing another PROMPT statement (such as PROMPT PG). By using this method, you can customize all the special keys on your keyboard to your liking. If you think this is useful, there's even more.

In addition to programming the keys by themselves, you can also program function key combinations such as Shift-F1. For example, the following command will redefine the Shift-F1 sequence to clear the screen with the CLS command.

```
prompt $e[0;84;"CLS$_"p
```

All the examples presented so far have automatically performed a carriage return for you. However, you may come across instances where you don't want a return. To avoid a carriage return, simply omit the $_ from your command. For example, the following prompt will assign the FORMAT command to the F3 key and not execute a return, thus allowing you to specify a drive letter.

```
prompt $e[0;61;"FORMAT "p
```

Notice the blank space just before the second quotation mark. This will output a blank space to the screen.

As you can see, the routines presented here can significantly reduce the number of keystrokes you must type while working with DOS. For your convenience, **Table 1** presents the sequence codes for all the function and cursor control keys (both alone and in combination with Control, Shift, and Alt). An n/a within the table means that particular key combination cannot be reprogrammed.

DEFINING THE KEYS AT STARTUP

If you like, you can create a batch file with a text editor to define your keys for you. The layout of your batch file would look something like this:

```
@echo off
cls
prompt $e[0;59;"dir$_"p
prompt $e[0;84;"cls$_"p
prompt $e[0;61;"format "p
.
additional function key definitions go here
.
prompt $p$g
```

In this simple listing, the first two statements suppress echoing to the screen and clear the screen. The function key definitions come next. Finally, the last statement resets the system prompt. If you want to enable these definitions each time you boot up, just insert that statements in AUTOEXEC.BAT or insert a statement in AUTOEXEC.BAT to call the appropriate batch file.

As you can see from the simple examples presented here, programming the special keys on your keyboard can really streamline your work with DOS. Have fun experimenting! ♦