STRUGGLING THROUGH

R. Roger Breton

This newsletter has been very long in coming. While the fault is of course mine, only part of that fault is directly mine. The major portion of the fault, I feel, lies with the fact that all except a very few of you seem to feel that this users' group is to be treated as a magazine subscription, wherein you pay a subscription fee and the publisher does the rest. For the umpteenth time, let me emphasize that this is not so!

TUG stands for "the TurboDOS Users' Group": a name and trademark most carefully chosen. Note that it is a "Users' Group," not a "User's Group." In both English and legalese the implication is that this is a group of, by, and for the users of TurboDOS collectively. I am again going to reiterate that it is not my intention to provide "Roger Breton's TurboDOS Newsletter." Were that my intention, it would be so named, a very tight issue schedule would be kept, and the price would be proportionate.

TUG is not dead, neither has it yet been mortally wounded, but it is certainly not well. Frankly, I am rapidly reaching the point where the lack of collective interest is destroying my interest. For all of you that have been with me from the beginning, let me state that if I don't receive enough input to fill the next issue before the next issue goes to bed, then the next issue, the fourth issue from me, may well be your last. At the very least, it will be the last at $20.00 per year.

For those of you who are new subscribers (since the last issue), rest assured that you will receive a total of four issues for your money, come what may.

Part of the delay in this issue was that I foolishly gave my mailing list to a highly trusted programmer to incorporate into a good data base last November. This individual got the programs almost done, then lost her system (literally). But not to fear! The entire mailing list was safe on tape backup! Unfortunately, a means of reading the tape was no longer available, and wasn't made available until the last week in June. This comedy of errors has caused several things to happen: first, never again will another individual, no matter how trusted, receive my only copy of anything; second, the almost done Volume 2:3 issue became outdated; third, in the interim I had completed a set of DO-file utilities to be commercially marketed; fourth, I needed something for this issue in a hurry; and fifth, I feel guilty that this issue has been so long in coming. The result, you are all receiving my DO-file utilities in their entirety in this issue, source code and all. Rest assured that your TUG membership fee is only a fraction of what this utility package was to have gone for commercially. Enjoy!

TABLE OF CONTENTS

<><><><><><><><><><><><><><><><><><><><><><><>

VERSIONS

R. Roger Breton

There has been considerable confusion about the various TurboDOS versions: what the differences are and whether or not a system should be updated. I hope in this very short article to dispell at least some of the myths that have arisen.

TurboDOS has been presented in the following versions: 1.00-1.16, 1.20-

VERSIONS -- Continued

1.22, 1.30, 1.40-1.43. The last version in each range was the "stable" debugged issue of that version (though there may be a series of patches required to fix the bugs). The current version is 1.43.

Versions 1.00 through 1.16 were the earlist versions of TurboDOS, created and issued when MP/M didn't work and MP/M-2 was a dream. As a result, a proprietary form of file locking, via the $.LOK pseudo-file, was used and record locking was not possible.

Versions 1.20 through 1.22 were the first MP/M-compatible versions as regards file and record locking. They still used extended BDOS function numbers to perform TurboDOS-only functions, however, and many of the newer CP/M and MP/M programs will not work because of this.

Version 1.30 eliminated the BDOS/TurboDOS function conflict, greatly improved overall performance, and introduced the ability to use 16-bit slaves.

Versions 1.40 and 1.41 introduced 16-bit masters, allowing a significant leap in performance, and the ability to network with IBM-PC's.

Version 1.42 greatly improved the networking capabilities of 1.41 and added several new functions.

Version 1.43 again improved networking capabilities, added more functions and increased the number of files that may be opened from a few hundred to well over a thousand, allowing increased power with database programs.

A system should always be upgraded to the most-recent version unless there has been a considerable investment in software that will not work under the newer version.

<><><><><><><><><><><><><><><><><><><><><><>

TUG'N, the TurboDOS Users' Group Newsletter, is published quarterly by the director of the TurboDOS Users' Group (TUG), and is available only to TUG members.

Membership in TUG is available at the following yearly rates: domestic or Canadian, $20.00; foreign, $30.00. Special consideration is given to contributors.

TUG'N is available on 8" SSSD CP/M or 5.25" 80-track TurboDOS standard format diskettes for a nominal fee. TUG'N is also available via modem at no charge under special circumstances and by appointment only. Write for details.

Back issues are available at the following rates: Volume 1:1, $7.50/10.00; Volume 2:1 or 2:2, $6.00/8.50.

 All correspondence should be addressed to:

R. Roger Breton, Director
TurboDOS Users' Group
836 Portola Avenue, Suite 599
Livermore, CA 94550

BATCH PROCESSING UNDER TurboDOS

R. Roger Breton

-- Introduction --

As most of us are probably aware, the TurboDOS operating system provides an extremely powerful tool in DO.COM, its batch processing utility. In a nutshell, DO.COM will process the commands and/or program input it finds in an ASCII auxilliary file, known as a DO-file, just as though they had been entered directly from the console keyboard at run-time. In this manner, routines that are long, repetitive, or tedious may be performed by the system with little or no human interaction.

What some may not realize, though it IS covered in the TurboDOS User's Guide, is that one or more arguments may be used to allow DO.COM to perform a task that has identical functions but different specifics, such as an edit-assemble-link-test process.

Associated with DO.COM is a utility, BATCH.COM, for background command processing via a dedicated slave processor in the system. This is accomplished through a special FIFO DO-file with the name BATCH.DO.

In addition to TurboDOS' own utilities, DO.COM and BATCH.COM, a series of sixteen auxiliary utilities specifically designed for use within a DO-file to allow DO.COM to alter the operations performed according to a series of internally or externally specified conditions is introduced. Since the key to these utilities is conditionality, DO-files incorporating them are known as conditional DO-files, and provide another level of utility and sheer computing power to the TurboDOS user. By their use, DO-files to meet virtually any reasnable condition may be created.

In order to minimize verbage and confusion, all utilities mentioned herein will be referred to as type COM. It should be clearly understood that type CMD utilities, for 16-bit systems, are also provided.

Also, since several of the command lines to be exampled here use either single or double quotes as part of the command line itself, all commands will be enclosed in angles <>, and will represent literal strings of the characters making up the command, plus the terminating carriage return. Standard TurboDOS conventions are used, wherein capital letters are used to represent explicit terms and lower-case letters are used to represent non-specific generic terms. Braces {} are used to enclose optional terms: the braces themselves are not part of the command.

-- The DO Command --

The batch processing utility DO.COM is executed via the following command line: <DO {uud:}filename{.typ} {arg1 arg2 ... argN}>. As with all TurboDOS command lines, this represents a sort of computer "sentence," with "DO" being the verb and "{uud:}filename{.type}" being the object. The string of arguments may be taken as a collection of modifiers.

The term "{uud:}filename{.typ}" is standard TurboDOS notation for a file representation, in this case the DO-file to be executed. The term "{uud:}"

represents an optional user/drive designation, such as the literal "4B:", in TurboDOS 1.4x format. For versions of turboDOS earlier than 1.4x, the "uud:" designation should be assumed to be "d:", as cross-user performance is not normally allowed except for global operation, even for priviledged users. Cross user operation is never allowed for non-priviledged users in any TurboDOS version, again except for global operations. If either the user code and/or drive letter is missing, then the current user and/or drive is assumed. The term "filename" represents any legal TurboDOS filename, and is NOT optional. The term "{.typ}" represents any legal TurboDOS filetype. If a filetype is not specified, type ".DO" is assumed.

The term "{arg1 arg2 ... argN}" represents an argument string of zero to "N" arguments. There may be any reasonable number of arguments in the command, all of which are separated from the DO-file and each other by white space -- spaces and/or tabs.

-- Simple DO-Files --

In its simplest form, a DO-file is nothing more than an ASCII text file containing a list of command lines identical to the form they would take were they entered from the keyboard at each running. This may be seen by studying a common example, a short file backup procedure using the COPY command with its ARCHIVED option. In this example, we are backing up the first thirty-one users of drive B: (hard-disk) to drive E: (floppy disk), keeping the same user numbers. We ourselves will be on drive A: user 0.

If we were to manually do this, the procedure would be as follows:

1) Enter COPY

2) Enter 0B: 0E:;NAC

3) Wait for the system to perform the task.

4) Enter 1B: 1E:;NAC

5) Wait for the system to perform the task.

-- Repeat 4) and 5) for users 2 through 29.

62) Enter 30B: 30E:;NAC

63) Wait for the system to perform the task.

64) Enter a carriage return.

And we are done! Except, of course, for any floppy disk changes required, which would have to be done in any case.

Had we created a DO-file, DAILY.DO, to do the work for us, our procedure would have become:

1) Enter DO DAILY

2) Wait for system to perform all the tasks.

DONE!

DAILY.DO would consist of the following:

```
COPY
0B: 0E:;NAC
1B: 1E:;NAC
2B: 2E:;NAC
...
28B: 28E:;NAC
29B: 29E:;NAC
30B: 30E:;NAC
<cr>
```

Notice that the commands are exactly as we would have entered them manually, including the final solo carriage return.

Creation of the DO-file is simplicity itself. Using any text editor or word processor, we would have simply entered our commands one at a time, just as we would have entered them had we been actually running them. If Wordstar or other word processor were used, it should have been used in the non-document mode.

-- Cascading DO-Files --

One DO-file may call another. The companion to DAILY.DO above is WEEKLY.DO. Were a weekly backup to be performed, the attributes of all files on user areas 0 through 31 of drive B: would have first been set to non-archived, then the normal daily backup would have been performed. This might have been done via the following DO-file, WEEKLY.DO:

```
SET
0B:;N-A
1B:;N-A
2B:;N-A
...
28B:;N-A
29B:;N-A
30B:;N-A
<cr>
DO DAILY
```

Please note that WEEKLY.DO ends with the command <DO DAILY>. Therefore, entering the command <DO WEEKLY> would have caused all the commands in WEEKLY.DO to be executed, including the execution of the command <DO DAILY>, which would have then caused all the commands in DAILY.DO to be executed! Both DO-files would be executed via a single command!

There is no fixed limit on the number of DO-files that may be cascaded, the limit depends upon the memory available for the DO-file command stack, and A.DO may call B.DO, which may call C.DO, which may call D.DO, etc. In a similar manner, A.DO may call B.DO and C.DO and D.DO from the same original file, or may call B.DO five times, or whatever.

There is only one restriction on the cascading of DO-files, and that is for a

DO-file calling itself. Since TurboDOS keeps track of which DO-file calls which, a DO-file calling itself will quickly build up a nest of pointers, and can easily get completely out of hand. Therefore, when a DO-file calls itself, the line doing the calling should NOT end with a carriage return (which is really a carriage return/linefeed pair), but rather end with a linefeed only. This will trick TurboDOS into NOT creating another stack entry for the same DO-file.

As an example of this in action, suppose we were to be creating a new utility, UTILITY.COM, and knew ahead of time that it would take many iterations until it was just right. We might create first the following DO-file, UTILITY.DO:

```
WM UTILITY.MAC
M80 UTILITY,UTILITY=UTILITY
GEN UTILITY.COM
UTILITY
<cr>
UTILITY
DO UTILITY<lf>
```

An analysis of the above shows that first we would enter the text editor WM.COM (Wordmaster) and edit the file UTILITY.MAC, which is our souce code for UTILITY.COM. Wordmaster takes its input ONLY from the keyboard (some other text editors do not), and will function perfectly in this context.

After editing our source code, we exit Wordmaster, whereupon the DO-file would promptly assemble our source code with the command M80 UTILITY,UTILITY=UTILITY, which would create the files UTILITY.REL and UTILITY.PRN.

The DO-file would then use GEN.COM, TurboDOS's own linker (and a very good one!) to link UTILITY.REL into UTILITY.COM. Since there is no UTILITY.GEN file, GEN.COM would go interactive, and would take the name of the REL-files to be linked from the DO-file. In this case, there is only one REL-file, UTILITY, followed by a solo carriage return to exit the interactive mode.

After UTILITY.COM has been linked, it would be tested by entering the command UTILITY. We would then note any possible errors in performance.

As expected, we would then be placed back into Wordmaster by having UTILITY.DO call itself. This process might be repeated indefinitely if the command <DO UTILITY> is NOT ended with a carriage return (a linefeed is used instead); TurboDOS would then keep only ONE stack entry, no matter how many iterations were made.

Once everything was finally correct with UTILITY.COM, we would exit from the DO-file by entering an attention-abort sequence, typically BREAK/control-C. We would then be left with UTILITY.MAC, our corrected source code; UTILITY.PRN, our assembly print file; UTILITY.REL, our relocatable object code module; and UTILITY.COM, our finished command file.

-- DO-File Program Input --

In all of our examples thus far, a command file was shown taking its input directly from the DO-file. Were we, in a normal manner, to enter the command line <COPY>, the command file COPY.COM would be loaded into memory and we would

find ourselves in interactive mode, operating from inside the command file itself (asterisk prompt) rather than at the operating system level (TurboDOS prompt, such as 0A}). We would then enter our sub-commands one at a time until we were finished, whereupon we would exit the interactive mode by entering a null sub-command (solo carriage return).

There is absolutely no difficulty encountered in making these entries from the DO-file. Simply remember to put everything in the DO-file in the exact order you would enter it directly: do not forget any solo carriage returns.

In the UTILITY.DO example above, we noted that Wordmaster took its input solely from the keyboard, whereas Wordstar and many other word processor and text editors would take their input from the DO-file. What makes this difference and how would we know? The difference as to whether a program will accept input from a DO-file, or will only accept input from the keyboard depends solely on how that program is written. Assuming the program was written in assembly language and it uses the standard console input functions, C-functions 2 and 10, as the input means, then it will also accept input from the DO-file. If, on the other hand, it uses the direct console I/O function, C-function 6 (and that only in the get combined status/input mode), as console input, then it will NOT accept input from a DO-file, and input MUST come from the console. Wordmaster falls into this latter category, while Wordstar is of the former type.

How can we tell into which category a given program will fall? Alas, there is no way other than to test the program in a DO-file. Fortunately, such a test is very rapid and easy to make.

-- DO-File Arguments --

Looking back at UTILITY.DO, we see it as it originally was:

```
WM UTILITY.MAC
M80 UTILITY,UTILITY=UTILITY
GEN UTILITY.COM
UTILITY
<cr>
UTILITY
DO UTILITY<1f>
```

Since the purpose of this DO-file was to edit, assemble, link and test a new utility, it would be much more useful if it would handle any new utility. We may do this by changing every "UTILITY" in the DO-file into an argument. Since the argument is always the same, they will all be the same argument. The new DO-file, which we'll call EALT.DO (after the initials of the functions involved), would now look like:

```
WM {1}.MAC
M80 {1},{1}={1}
GEN {1}.COM
{1}
<cr>
{1}
DO {1}<1f>
```

Assuming that we are still working on the program UTILITY.COM, we would execute

our new DO-file with the command <DO EALT UTILITY>. In the command line, "DO" is the command itself, "EALT" is the name of the DO-file to be executed, and "UTILITY" is the first (and only) argument, which will replace every occurance of "{1}", the first argument place marker, in the DO-file.

In order to do this replacing, a temporary DO-file, EALT.DO$, is created on-the-fly and contains all the replacements. It is this temporary DO$-file that is executed. The DO$-file is deleted after execution, unless, of course, you attention-abort out, as with our example, in which case we would have to manually delete it.

Any reasonable number of arguments may be used. Looking at our DAILY.DO example above, we may increase the flexibility to any hard-disk drive, A:, B:, C: or D:, backed up to either floppy drive, E: or F:, by making the DO-file contain two arguments:

```
COPY
0{1}: 0{2}:;NAC
1{1}: 1{2}:;NAC
2{1}: 2{2}:;NAC
...
28{1}: 28{2}:;NAC
29{1}: 29{2}:;NAC
30{1}: 30{2}:;NAC
<cr>
```

To use this DO-file to back up hard-disk drive C: to floppy drive F:, we would execute the command <DO DAILY C F>.

It is important for us to remember that the substituted arguments are truly substituted, and that argumented DO-files are cascaded as easily as any others if planned properly. Witness our example WEEKLY.DO file converted to arguments:

```
SET
0{1}:;N-A
1{1}:;N-A
2{1}:;N-A
...
28{1}:;N-A
29{1}:;N-A
30{1}:;N-A
<cr>
DO DAILY {1} {2}
```

By making the <DO DAILY> command <DO DAILY {1} {2}>, we will cascade the arguments into the DAILY.DO file.

--Arguments with Spaces --

Normally, the arguments are separated by spaces in the command line. Occasionally, some argument may appear that in itself requires spaces. Suppose the DO-file RUNBAS.DO contains the command line <MBASIC {1}>. The argument would usually be the name of a file of type BAS. An execution command might be <DO RUNBAS AR> where "AR.BAS" is the name of an MBASIC accounts receivable program.

Let us assume, however, that the AR.BAS file itself has some argument or switch involved, such as "$512" for 512-byte records. In this case, the execution command should be <DO RUNBAS AR $512>, but this won't work, as only "AR" would be taken as the first argument, and "$512" would be taken as the second argument and, since no second argument is called for, "$512" would be ignored.

The solution is both simple and elegant: enclose the argument in single or double quotes! The correct command line would be either <DO RUNBAS 'AR $512'> or <DO RUNBAS "AR $512">. Why either single or double quotes? In case the argument called for should contain quotes within itself: in the command <DO DOFILE '"TEST"'> the argument is "TEST" (quotes included), not TEST.

-- Defaulted DO-File Arguments --

When creating argumented DO-files, we often run across the situation where the argument is normally a specific value, and only occasionally a differing one. In this case, we may use defaulted arguments. Using our trusty DAILY.DO again, let us assume that almost all our work is done on hard-disk drive B:, and that we almost always back up to floppy drive E:: The other hard-disk drives are for special use, seldom changed (hence seldom backed up), and floppy drive F: is a 5-1/4" floppy only rarely used. In this case, we could make our DAILY.DO file look like:

```
COPY
0{1,B}: 0{2,E}:;NAC
1{1,B}: 1{2,E}:;NAC
2{1,B}: 2{2,E}:;NAC
...
28{1,B}: 29{2,E}:;NAC
29{1,B}: 29{2,E}:;NAC
30{1,B}: 30{2,E}:;NAC
<cr>
```

We now have maximum flexibility. Should we wish to backup B: to E: (normal), we simply enter the command <DO DAILY ''>. The '' is two single-quotes togther, and causes a NULL ARGUMENT to be entered. A null argument may be either two single-quotes or two double-quotes. A null argument is still an argument, and AT LEAST ONE ARGUMENT MUST BE ENTERED if defaulted arguments are used. This triggers TurboDOS to make the DO$-file required. The command <DO DAILY> is NOT acceptable, as no DO$-file would be created and TurboDOS would try to run the DO-file as it stands.

In a like manner, if we wished to backup drive D: to drive E:, the command could be <DO DAILY D>. Since "D" is a valid argument, the DO$-file will be created and the second argument will assume its defaulted value.

To backup drive C: to drive F: the command <DO DAILY C F>, with both arguments, is required, but to backup drive B: to drive F:, either <DO DAILY B F> or <DO DAILY "" F> may be used. The former uses explicit arguments, while the latter uses a null argument for the first argument, causing the default, "B", to be substituted.

-- Background Batch Processing --

TurboDOS has a very simple yet elegant method of using its DO.COM utility to perform time-consuming operations not requiring human intervention as a completely invisible background process. This is through the use of a dedicated processor, the special FIFO DO-file BATCH.DO, and the utility BATCH.COM.

For most systems, the key issue here is the dedicated processor. Before throwing up your hands, however, and saying that hardware costs prohibit using backgroud batch processing, please bear in mind that time consuming processes, such as a Dataflex re-index operation or some large sort routine, may take hours to run, and represent lost time on the terminal upon which they are done. The cost of an additional slave processor might well be quickly realized in the savings of lost operator time. Even some relatively trivial function such as copying floppies onto the hard disk take considerable operator time, and can be better relegated to a background process.

To set up the background processor, install a slave with its own OSSLAVEx.SYS file, patching its AUTUSR parameter to 80 so it will automatically boot priviledged to user 0 of drive A:, and its WARMFN to 00,"BATCH ","AUT" so it will warmstart to the file BATCH.AUT. A console on the slave is not required. In fact, since it is priviledged, a console should be avoided.

Create BATCH.AUT with the command line <AUTOLOAD |DO BATCH\RENAME AUTOLOAD.AUT BATCH.AUT\SET BATCH.AUT;GR>. Note that there are really three commands in this command line, and that the new file BATCH.AUT is set global and read-only.

The next step is to create the special FIFO DO-file BATCH.DO. A FIFO (First-In-First-Out) file is a very special file type supported by TurboDOS, having always a fixed number of records set at the time the file is created and accessing those records on a strictly first-in-first-out basis. A FIFO file is somewhat like a Unix "pipe," and is much like placing marbles in a real pipe. The first marble in is the first out the other end. The pipe will be able to contain a specific number of marbles at one time, which is analogous to the length (number of records) of a FIFO.

The subject of FIFO files cannot be covered in detail here, but that shouldn't stop us from being able to create BATCH.DO and successfully use it. We create BATCH.DO with the command <FIFO BATCH.DO>, then answer the following questions as they appear on the screen (our entries are underlined):

```
FIFO file not found, creating new file
Enter FIFO type (Ram/Disk): D
Suspend processing on full/empty (Y/N): Y
Enter maximum number of records (1-65535): 200
FIFO file created
```

Taking each question in turn, the body of a FIFO-file may be RAM- (memory-) resident or Disk-resident. RAM-resident FIFO's are much faster, but limited to a maximum of 127 records (usually much less), and the benefits of RAM-resident FIFO's are not usually realized in BATCH.DO.

A FIFO is "empty" when none of its records contain current data, and "full" when all contain current data. When a fifo is empty, it cannot be read (nothing left to read), and when it is full it cannot be written (no room to write). Under

such conditions, attempting to access the FIFO will produce either a suspension of operation until the read or write can take place or an error message will be returned dropping the processor to the operating system level. Since we have no console on our processor, we need to suspend operation.

The size or length of a FIFO-file is counted in records. Each record of BATCH.DO represents one command line that may be "piped" into the FIFO. Command lines will be written in at one end and read out of the other in a steady stream, therefore, there should be enough records allocated to allow the largest desired "queue" of command lines to be contained. The length of time for any single command line to be processed is, of course, dependent upon the operations to be performed.

Please bear in mind that we are speaking of command lines here, not commands. One command line may contain many commands, and a single command may be the execution of a DO-file consisting of any reasonable number of commands.

After BATCH.DO has been created, it must be located on user area 0 of drive A: and must be set global. Do NOT set it read-only.

After rebooting, the dedicated slave will attempt to process BATCH.DO, which will be empty. The dedicated slave will then suspend operation (wait) until a command line is written to BATCH.DO, whereupon it will immediately process the command line, then wait for the next.

Command lines are written to BATCH.DO from anywhere on the system by the BATCH.COM utility. You need not be priviledged to send a command line to BATCH.DO. As an example, the command line <BATCH SORT FILA.DAT|SORT FILB.DAT|SORT FILC.DAT>, when executed from user 5 of drive C:, sends BATCH.DO this command line <5C:\SORT FILA.DAT\SORT FILB.DAT\SORTC.DAT>. Notice that the first command is <5C:>, which will immediately move the dedicated slave's operation to the local drive and user, so the operator need not be priviledged and so it will operate on the local files, just as the operator would. Also notice that all vertical bars "|" have been converted to backslashes "\", TurboDOS' standard command separator.

For those who need more than one background process, it is very easy to modify BATCH.COM for files other than BATCH.DO. To make a "BATCH1" system, copy BATCH.COM to BATCH1.COM, use the MONITOR.COM utility to modify BATCH1.COM as follows:

Enter MONITOR

Enter LBATCH1.COM

Enter W00 42 41 54 43 48 20 20 20 44 4F 20 00 00 00 00

This will return an address "nnnn". Enter Ennnn

Respond to the "E" command as follows:

nnnn: 00 = <cr>
nnnn: 42 = <cr>
nnnn: 41 = <cr>
nnnn: 54 = <cr>

```
nnnn: 43 = <cr>
nnnn: 48 = <cr>
nnnn: 20 = 31<cr>
nnnn: 20 = <esc>
```

Enter SBATCH1.COM

Exit MONITOR by entering Q

BATCH1.COM has been modified. Create the background process exactly as previously described, substituting "BATCH1" for every occurance of "BATCH". Use BATCH1.COM to access the processor.

-- Conditionality in DO-Files --

What is meant by conditionality in DO-files? Simply speaking, a DO-file is conditional if it changes function or operation according to the conditions that exist at the time of execution of the various commands within the the DO-file. A considerable number of TurboDOS's utilities are conditional in some form. The command <COPY B: E:;NAC> will copy each of the files on drive B: to drive E: IF AND ONLY IF the Archived attribute on that file is set. In other words, only those files with the Archived attribute set will be copied. This is clearly a conditional operation.

Because of this, our DAILY.DO and WEEKLY.DO examples above may be said to be conditional DO-files, and truly are. If this is the case, however, why is so much of this article devoted to conditionality and what is all this brouhaha about conditional DO-files and the conditional utilities in the first place? The difference is in implicit and explicit conditionality.

A large number of TurboDOS utilities, as stated above, have some implicit conditionality. The conditional utilities presented here, however, were specifically written to provide conditionality in a direct, explicit manner.

-- Conditional Utilities --

There are sixteen utilities presented here: six were created as explicit conditionals and ten were created to support the six conditionals, though two are so basic to use with DO-files in general that they could hardly be classed as conditional support utilities.

The early forms of these utilities were presented as part of my now-defunct TurboTOOLS utility/module set, and were so popular that they have been considerably expanded, improved and updated and are here in version 2.20. All are presented in both 8- and 16-bit versions, and will operate under TurboDOS version 1.22 and later.

The six explicitly conditional utilities are divided into three pairs: one pair, IFFIL and IFNOTFIL, for the execution of a command string based upon the presence of a specified file; one pair, IFUSR and IFNOTUSR, for the execution of a command string based upon the presence of files on a given user area of a given drive; and one pair, IFCHR and IFNOTCHR, for the execution of a command string based upon the matching of an argument. The ten support utilities are divided into three pairs and four singles: one pair, CRFIL and CRFILYES, for the creation of dummy (zero-K) files for use by IFFIL and IFNOTFIL; one pair,

DLFIL and DLFILYES, for the deletion of dummy files; one pair, ENDDO and ENDDOYES, for the premature termination of a DO-file; one single, DOHALT, to force the halting of the operation of a DO-file to allow human interaction; one single, PROMPT, for the display of a special message file on the console screen; one single, CLS, for the clearing of the console screen, and one single, BEEP, used to beep the console three times as an operator-alert signal. Each of these sixteen utilities is described in detail later.

-- Basic Command Syntaxes --

As would be expected in an integrated package, the command syntax for each of the sixteen utilities is related. The command syntaxes are:

```
IFFIL testfile commandlist

IFNOTFIL testfile commandlist

IFUSR uud: commandlist

IFNOTUSR uud: commandlist

IFCHR tchr mchr commandlist

IFNOTCHR tchr mchr commandlist

CRFIL zerofile

CRFILYES zerofile {;promptstring}

DLFIL zerofile

DLFILYES zerofile {;promptstring}

ENDDO

ENDDOYES {;promptstring}

DOHALT {;promptstring}

PROMPT dispfile

CLS

BEEP
```

-- Designating File Names --

As can be readily seen, a majority of the utilities specify some file be entered, either "testfile", "zerofile" or "dispfile". All of these are to be entered in the standard TurboDOS file name format of "{uud:}filename{.typ}". The filename is not optional and must be explicit: wildcards, "?" or "*", are not allowed. The user area/drive code designator and/or filetype are optional, with the current user area, current drive and/or a null filetype assumed as default.

A note about the user area/drive code designator "uud:": this is the TurboDOS 1.4x designator, and should be taken to mean "d:" for versions 1.22 and 1.30. As a result, cross-user performance is not allowed except in TurboDOS 1.4x. This is because 1.4x has a quite clever method of parsing the user code upon which these utilities depend. As is always the case, "{uud:}" may be taken to mean that either the user area or the drive or both are optional. The current user area and/or drive will be understood as default for any missing parameter.

It must be pointed out that IFUSR and IFNOTUSR require BOTH the user code (except for versions 1.22 and 1.30) AND the drive to operate. If either is missing, an error will occur.

-- The Command List --

The six explicit conditional utilities require a command list "commandlist" to operate, if "commandlist" is missing, an error will occur. "Commandlist" may be a list of one or more commands in the form <{|}command {|command {|command {...}}}> where "command" is any legitimate TurboDOS command, with all required options, etc. The vertical bar "|" is used as a command separator, and is converted to the conventional backslash "\" prior to execution. In version 1.4x only, if the command list begins with a command separator, then the commands will not be echoed to the console as they are run.

Provisions have been made for the nesting of multiple IFFIL, IFNOTFIL, IFUSR, IFNOTUSR, IFCHR and IFNOTCHR command strings: the first (outer) nest must have a single separator character, "|"; the second nest a double separator character, "||"; the third nest a triple separator character, "|||"; etc. The following example would process as shown:

```
      1         2    3         4     5         6      7
      |         |    |         |     |         |      |
   0A}IFCHR A A BEEP|IFCHR B B BEEP||IFCHR C C BEEP|||BEEP
```

Breaking this down, the primary command at #1, IFCHR, will run the following command line:

```
          2    3         4     5         6      7
          |    |         |     |         |      |
   0A}    BEEP\IFCHR B B BEEP |IFCHR C C BEEP ||BEEP
```

Notice that the single special separator "|" has become a standard TurboDOS separator "\", the double special separator "||" has become a single " |", and the triple special separator "|||" has become a double " ||". In actual fact, the leading special separator in any muliple group will have been replaced by a space.

Notice also that command #2 is separated from the "0A}" prompt by four spaces. These four spaces are an overwrite of the "A A " parameters, and are a result of the method I chose to process the command string. Since leading spaces have no effect on a command, command #2 will operated as desired.

Running the entire line will produce the operation depicted below, the dash-numbers at the extreme right being the number of the command being executed:

```
   0A}IFCHR A A BEEP|IFCHR B B BEEP||IFCHR C C BEEP|||BEEP                --1
```

```
0A}    BEEP                                                    --2

0A}IFCHR B B BEEP |IFCHR C C BEEP ||BEEP                       --3

0A}    BEEP                                                    --4

0A}IFCHR C C BEEP  |BEEP                                       --5

0A}    BEEP                                                    --6

0A}BEEP                                                        --7
```

The maximum length of the command line, counting all spaces, is 126 characters after the initial command. To emphasize this, graphically the maximum length of the command line is depicted below:

```
          |<----- 126 characters absolute maximum ----->|
          |                                             |
0A}IFCHR A A BEEP|IFCHR B B BEEP||IFCHR C C BEEP|||BEEP
```

The reason for this is that that 126 characters will comprise the command tail for the original command, and this command tail must fit into the 128-byte default DMA buffer at address 0080 (DS:0080 for 16-bit), with its actual length as a leading byte and an ACSII null as a trailing byte. A longer command list will be truncated by TurboDOS.

-- The Promptstring --

Four of the conditional-support commands, CRFILYES, DLFILYES, ENDDOYES and DOHALT, allow an optional "promptstring" to be entered. Each of these four commands cause a string of text, the prompt string, to be displayed, and will halt the DO-file until a keyboard input is provided. With the exception of the DOHALT file, any character except a "Y" or "y" will be assumed to be an "N", and the specified action will not occur, allowing the DO-file to continue unchanged.

The prompt string displayed by these four commands will be a default prompt peculiar to each command, unless the optional "promptstring" is specified. "Promptstring" may be any sting of printable ASCII characters except the backslash "\", which TurboDOS would interpret as the end of the command, and the dollar sign "$", which the utility would interpret as the end of "promptstring". Due to the way in which TurboDOS parses commands, all lower-case letters will be converted to upper case. Note that "promptstring" is considered an option, and must start with a semicolon ";".

Two special characters are designated for use in the first position, or, if both special characters are used, the first two positions after the semicolon. These special characters are the circumflex "^", which will cause the console screen to clear, and the asterisk "*", which will cause the console to NOT beep (a beep is the standard default). If these are the only characters in "promptstring" the default prompt will be displayed with screen clearing and/or console silencing. This may be best depicted in table form, where "xxxxx" represents the text of "promptstring":

Promptstring	Clear	Bell	Prompt
None or ;	No	Yes	Default
;^	Yes	Yes	Default
;*	No	No	Default
;^* or ;*^	Yes	No	Default
;xxxxx	No	Yes	Promptstring
;^xxxxx	Yes	Yes	Promptstring
;*xxxxx	No	No	Promptstring
;^*xxxxx or ;*^xxxxx	Yes	No	Promptstring

It is very important to watch your chracters carefully, especially trailing spaces or tabs, as ";^*" will clear the screen and display the default prompt without sounding the bell, while ";^* " will clear the screen and display a space without sounding the bell. The space is not visible, therefore it is an effective null prompt.

The maximum length of "promptstring" is 76 characters, not counting the semicolon, the circumflex or the asterisk. Please bear in mind that if the command using "promptstring" is itself part of "commandlist" discussed previously, the 126-byte total length still applies.

-- Utility Pairing --

As has been mentioned, most of these utilities are paired: IFFIL and IFNOTFIL; IFUSR and IFNOTUSR; IFCHR and IFNOTCHR; CRFIL and CRFILYES; DLFIL and DLFILYES; and ENDDO and ENDDOYES. This pairing is accomplished by having both members of a pair identical except for one patchable byte. When this byte is 00 hex, the first member of the pair is chosen, and when not 00 hex, the second member. This patchable byte is always located at address 0103 for 8-bit utilities and DS:0100 for 16-bit. A table of the patches is:

Patch = 00	Patch <> 00
IFFIL	IFNOTFIL
IFUSR	IFNOTUSR
IFCHR	IFNOTCHR
CRFIL	CRFILYES
DLFIL	DLFILYES
ENDDO	ENDDOYES

By the use of this pairing technique a great reduction in the number of source code files is achieved, as well as a simplification of the linking process.

-- Console Screen Clearing --

The utilities CRFILYES, DLFILYES, ENDDOYES, DOHALT, PROMPT and CLS have the ability to clear the console screen. This is accomplished via a twelve-byte-maximum string of characters in a special patch area of the utilities. This twelve-byte string consists of the characters necessary to home the cursor and clear the screen, with the remaining bytes padded with dollar signs (code 24 hex, 36 decimal). The default is a control-Z (code 1A hex, 26 decimal) and eleven dollar signs, as this sequence will clear most terminals. Patch the console clear-screen strings to the codes required by your terminals.

The locations of the first byte of the clear-screen strings for the above utilities are address 0104 for 8-bit utilities and DS:0101 for 16-bit.

-- Conditional Utility Operation --

The file-conditional utilities, IFFIL and IFNOTFIL, have been created to execute a command list based upon the presence of a file. The syntaxes and operations of these utilities are:

IFFIL testfile commandlist

"Commandlist" will be executed IF AND ONLY IF "testfile" exists.

IFNOTFIL testfile commandlist

"Commandlist" will be executed IF AND ONLY IF "testfile" does not exist.

The user-conditional utilities, IFUSR and IFNOTUSR, have been created to execute a command list based upon the assignment of files to a specific user area of a specific drive. The syntaxes and operations of these utilities are:

IFUSR uud: commandlist

"Commandlist" will be executed IF AND ONLY IF there are files assigned to user area/drive "uud:".

IFNOTUSR uud: commandlist

"Commandlist" will be executed IF AND ONLY IF there are no files assigned to user area/drive "uud:"

The argument-conditional utilities, IFCHR and IFNOTCHR, have been created to execute a command list based upon matching of a test character to a character entered into the DO-file as an argument. The syntaxes and operations of these utilities are:

IFCHR tchr mchr commandlist

"Commandlist" will be executed IF AND ONLY IF "tchr" is identical to "mchr".

IFNOTCHR tchr mchr commandlist

"Commandlist" will be executed IF AND ONLY IF "tchr" is not identical to "mchr".

The test character, "tchr", and the match character, "mchr", may be any single printable ASCII character legal in a file name, as TurboDOS' filename parsing facility is used. Normally, an alphanumeric character should be used and punctuation marks avoided. Lower-case characters are automatically converted to upper-case by TurboDOS. If more than one character is entered for either "tchr" or "mchr", only the first character is relevant.

In normal use, "tchr" is directly entered and "mchr" is indirectly entered, as in the DO-file command line <IFCHR D {2} |COPY B:*.DAT;NCA|COPY B:*.MAS;NCA>. The command list "|COPY B:*.DAT;NCA|COPY B:*.MAS;NCA" will be converted to "\COPY B:*.DAT;NCA\COPY B:*.MAS;NCA" and executed IF AND ONLY IF the second DO-

file argument is "D". In actual practice, "tchr" and "mchr" are completely interchangable. The command <IFCHR {2} D |COPY B:*.DAT;NCA|COPY B:*.MAS;NCA> is identical to the previous one.

-- Support Utility Operation --

The support utilities consist of two files to create a dummy file, two to delete a dummy file, two to terminate the DO-file, one to halt the DO-file, one to display a special prompt file, one to beep the console, and one to clear the console screen.

The dummy-file creation utilities, CRFIL and CRFILYES, will create a dummy or zero-K file (one that has no contents). While the principal use for a dummy file is as a test file for IFFIL or IFNOTFIL, it is also suitable for use wherever an empty file is required, as in a clean SYSLOG.SYS file to use with the log-on/log-off procedures. Should a file of the selected name already exist at that user area/drive, or if for any reason a file creation cannot take place an error will be displayed and the utility aborted. The syntaxes and operations of these utilities are:

CRFIL zerofile

"Zerofile" will be created.

CRFILYES zerofile {;promptstring}

"Zerofile" will be created IF AND ONLY IF the response to "promptstring" is "Y". If "promptstring" is not specified, then the default prompt of "Okay to create (zerofile) at this time?" will be used.

The dummy-file deletion utilities, DLFIL and DLFILYES, will delete a dummy or zero-K file. Should a file of the selected name not already exist at that user/drive, should it not be empty, should it be set read-only or FIFO, or if for any reason a file deletion cannot take place an error will be displayed and the utility aborted. The syntaxes and operations of these utilities are:

DLFIL zerofile

"Zerofile" will be deleted.

DLFILYES zerofile {;promptstring}

"Zerofile" will be deleted IF AND ONLY IF the response to "promptstring" is "Y". If "promptstring" is not specified, then the default prompt of "Okay to delete (zerofile) at this time?" will be used.

Please note that the DLFIL and DLFILYES utilities are completely different from the TurboDOS utility DELETE.COM, as they will only act upon a dummy (zero-K) file.

The DO-file termination utilities will cause a termination of the DO-file, and all pending or stacked DO-files, running on that processor. There is no effect on DO-files on other processors. These utilities may be used to abort a DO-file as the result of a conditional judgement. The syntaxes and operations of these utilities are:

ENDDO

Terminate all pending and stacked DO-files.

ENDDOYES {;promptstring}

Terminate all pending and stacked DO-files IF AND ONLY IF the response to "promptstring" is "Y". If "promptstring" is not specified, the default prompt of "Okay to terminate DO-file at this time?" will be used.

The DO-file operation suspension utility, DOHALT, will cause the operation of a DO-halt to stop and wait for a console input. The syntax and operation of this utility is:

DOHALT {;promptstring}

"Promptstring" is displayed and the DO-file is halted pending keyboard input. If "promptstring" is not specified, the default prompt of "DO-file halted, press any key to continue." will be used.

The prompt-file display utility, PROMPT, will cause a the contents of a special prompt-file to be displayed. The syntax and operation of this utility is:

PROMPT dispfile

The contents of "dispfile" are displayed. "Dispfile" is an ordinary text file with a few special features and limitations. It is perhaps easiest to think of this file as a sort of super-promptstring, as the circumflex and/or asterisk are used in the identical manner (first two bytes of the file) to provide screen clearing and/or suppress the bell. The limitations are basically length and function: the file may not exceed 2048 bytes (2K) total length, and to be functional must not contain more lines of data than can be displayed on the console.

The screen-clearing utility, CLS, simply clears the screen. Entering the command <CLS> is the only mode of operation.

The operator-alert utility, BEEP, is also a single-mode command utility, the command being <BEEP>. When used, this utility will make the console beep three times at about one-half second intervals. This beeping provides a clear and distinct signal to alert the operator that interaction is required or that the DO-file is finished.

-- A Simple Working Example --

As a simple working example, let's make a trivial BACKUP.DO file which may be used for either backing-up or restoring in a system with four logical hard-disk drives and two floppy drives. The secret to this file depends on the knowledge that the hard-disk drives are drives A:, B:, C: or D: and that the floppy-disk drives are E: and F:. Whether we are backing-up or restoring, therefore, may be determined by comparing the desired drives.

Our finished DO-file will be normally executed via the simple command <DO BACKUP d1 d2 {W}> where "d1" is the source drive, "d2" is the destination drive and "W"

is the WEEKLY BACKUP switch, which is valid only if a backup, rather than a restore, is being done. Our DO-file is created as follows:

Determine if we have a BACKUP or a RESTORE: if neither, abort out.

```
IFCHR A {1} CRFIL B.$
IFCHR B {1} CRFIL B.$
IFCHR C {1} CRFIL B.$
IFCHR D {1} CRFIL B.$
IFCHR E {1} CRFIL R.$
IFCHR F {1} CRFIL R.$
IFNOTFIL B.$ IFNOTFIL R.$ ENDDO
...
```

If we have a BACKUP, is it legal? If no, abort out.

```
...
IFFIL B.$ IFCHR E {2} CRFIL T.$
IFFIL B.$ IFCHR F {2} CRFIL T.$
IFFIL B.$ IFNOTFIL T.$ DLFIL B.$|ENDDO
DLFIL T.$
...
```

Same thing for a RESTORE.

```
...
IFFIL R.$ IFCHR A {2} CRFIL T.$
IFFIL R.$ IFCHR B {2} CRFIL T.$
IFFIL R.$ IFCHR C {2} CRFIL T.$
IFFIL R.$ IFCHR D {2} CRFIL T.$
IFFIL R.$ IFNOTFIL T.$ DLFIL R.$|ENDDO
DLFIL T.$
...
```

Do the actual BACKUP or RESTORE for all 32 users by calling a subordinate DO-file BACKUPX.DO.

```
...
IFUSR 0{1}: DO BACKUPX {1} {2} {3,@} 0
IFUSR 1{1}: DO BACKUPX {1} {2} {3,@} 1
IFUSR 2{1}: DO BACKUPX {1} {2} {3,@} 2
...  (Repeat command line for users 3 through 28)
IFUSR 29{1}: DO BACKUPX {1} {2} {3,@} 29
IFUSR 30{1}: DO BACKUPX {1} {2} {3,@} 30
IFUSR 31{1}: DO BACKUPX {1} {2} {3,@} 31
...
```

Clean up after ourselves.

```
...
IFFIL B.$ DLFIL B.$
IFFIL R.$ DLFIL R.$
BEEP
```

Our subordinate DO-file, BACKUPX.DO, consists of:

```
IFFIL B.$ IFCHR W {3} SET {4}{1}:*.*;N-A
IFFIL B.$ COPY {4}{1}: {4}{2}:;NAC
IFFIL R.$ COPY {4}{1}: {4}{2}:;NX
```

By the use of these DO-files, a somewhat complex and tedious task has been reduced to a single command.

-- A Complex Working Example --

To provide ourselves with a more exhaustive working example, let's design a different BACKUP.DO file to form the core of a more comprehensive backup system. In this DO-file, the computer will ask us a series of yes/no questions, and produce a BACKUP or RESTORE based upon our answers.

Create a nice sign-on screen, BACKUP.MSG.

Use BACKUP.DO to display this message and give us a chance to abort gracefully.

```
\PROMPT BACKUP.MSG\CRFILYES T.$;*DO YOU WISH TO CONTINUE?
IFNOTFIL T.$ ENDDO
DLFIL T.$
...
```

Notice the leading separator character "\" in the first command line, this will cause the command <CRFILYES T.$;*DO YOU WISH TO CONTINE?> to not be itself echoed to the console (in version 1.4x) and will produce a better-looking output.

Are we going to BACKUP or RESTORE?

```
...
CRFILYES R.$;^RESTORE (Y) OR BACKUP (N) FILES?
...
```

Assuming a BACKUP, is it WEEKLY or DAILY?

```
...
IFNOTFIL R.$ CRFILYES W.$;WEEKLY (Y) OR DAILY (N) BACKUP?
...
```

Assuming a RESTORE, restore only missing files or all files?

```
...
IFFIL R.$ CRFILYES M.$;MISSING-ONLY (Y) OR ALL (N) FILES?
...
```

Backup from which drive(s)?

```
...
CRFIL T.$
IFNOTFIL R.$ CRFILYES A.$;FROM DRIVE A (Y/N)?|DLFIL T.$
IFNOTFIL R.$ CRFILYES B.$;FROM DRIVE B (Y/N)?|DLFIL T.$
IFNOTFIL R.$ CRFILYES C.$;FROM DRIVE C (Y/N)?|DLFIL T.$
IFNOTFIL R.$ CRFILYES D.$;FROM DRIVE D (Y/N)?|DLFIL T.$
```

```
     IFNOTFIL R.$ IFFIL T.$ IFFIL W.$ DLFIL W.$|DLFIL T.$|ENDDO
     ...
```

Restore from which drive?

```
     ...
     IFFIL R.$ CRFILYES E.$;FROM DRIVE E (Y/N)?|DLFIL T.$
     IFFIL T.$ IFFIL R.$ CRFILYES E.$;FROM DRIVE E (Y/N)?|DLFIL T.$
     IFFIL T.$ DLFIL R.$|IFFIL M.$ DLFIL M.$|DLFIL T.$|ENDDO
     ...
```

Restore to which drive?

```
     ...
     CRFIL T.$
     IFFIL R.$ CRFILYES A.$;TO DRIVE A (Y/N)?|DLFIL T.$
     IFFIL T.$ IFFIL R.$ CRFILYES B.$;TO DRIVE B (Y/N)?|DLFIL T.$
     IFFIL T.$ IFFIL R.$ CRFILYES C.$;TO DRIVE C (Y/N)?|DLFIL T.$
     IFFIL T.$ IFFIL R.$ CRFILYES D.$;TO DRIVE D (Y/N)?|DLFIL T.$
     IFFIL T.$ IFFIL R.$ IFFIL E.$ DLFIL E.$|IFFIL F.$ DLFIL F.$|IFFIL M.$
          DLFIL M.$|DLFIL R.$|DLFIL T.$|ENDDO
     ...
```

Backup to which drive?

```
     ...
     IFNOTFIL R.$ CRFILYES E.$;TO DRIVE E (Y/N)?|DLFIL T.$
     IFFIL T.$ IFNOTFIL R.$ CRFILYES F.$;TO DRIVE F (Y/N)?|DLFIL T.$
     IFFIL T.$ IFFIL A.$ DLFIL A.$|IFFIL B.$ DLFIL B.$|IFFIL C.$ DLFIL C.$
          |IFFIL D.$ DLFIL D.$
     IFFIL T.$ IFFIL W.$ DLFIL W.$|DLFIL T.$|ENDDO
     ...
```

Tidy things up.

```
     ...
     IFNOTFIL R.$ IFNOTFIL W.$ IFFIL A.$ IFFIL E.$ CRFIL DAE.$
     IFNOTFIL R.$ IFNOTFIL W.$ IFFIL A.$ IFFIL F.$ CRFIL DAF.$
     IFNOTFIL R.$ IFNOTFIL W.$ IFFIL B.$ IFFIL E.$ CRFIL DBE.$
     IFNOTFIL R.$ IFNOTFIL W.$ IFFIL B.$ IFFIL F.$ CRFIL DBF.$
     IFNOTFIL R.$ IFNOTFIL W.$ IFFIL C.$ IFFIL E.$ CRFIL DCE.$
     IFNOTFIL R.$ IFNOTFIL W.$ IFFIL C.$ IFFIL F.$ CRFIL DCF.$
     IFNOTFIL R.$ IFNOTFIL W.$ IFFIL D.$ IFFIL E.$ CRFIL DDE.$
     IFNOTFIL R.$ IFNOTFIL W.$ IFFIL D.$ IFFIL F.$ CRFIL DDF.$
     IFNOTFIL R.$ IFFIL W.$ IFFIL A.$ IFFIL E.$ CRFIL WAE.$
     IFNOTFIL R.$ IFFIL W.$ IFFIL A.$ IFFIL F.$ CRFIL WAF.$
     IFNOTFIL R.$ IFFIL W.$ IFFIL B.$ IFFIL E.$ CRFIL WBE.$
     IFNOTFIL R.$ IFFIL W.$ IFFIL B.$ IFFIL F.$ CRFIL WBF.$
     IFNOTFIL R.$ IFFIL W.$ IFFIL C.$ IFFIL E.$ CRFIL WCE.$
     IFNOTFIL R.$ IFFIL W.$ IFFIL C.$ IFFIL F.$ CRFIL WCF.$
     IFNOTFIL R.$ IFFIL W.$ IFFIL D.$ IFFIL E.$ CRFIL WDE.$
     IFNOTFIL R.$ IFFIL W.$ IFFIL D.$ IFFIL F.$ CRFIL WDF.$
     IFFIL R.$ IFFIL E.$ IFFIL A.$ CRFIL REA.$
     IFFIL R.$ IFFIL E.$ IFFIL B.$ CRFIL REB.$
     IFFIL R.$ IFFIL E.$ IFFIL C.$ CRFIL REC.$
```

```
IFFIL R.$ IFFIL E.$ IFFIL D.$ CRFIL RED.$
IFFIL R.$ IFFIL F.$ IFFIL A.$ CRFIL RFA.$
IFFIL R.$ IFFIL F.$ IFFIL B.$ CRFIL RFB.$
IFFIL R.$ IFFIL F.$ IFFIL C.$ CRFIL RFC.$
IFFIL R.$ IFFIL F.$ IFFIL D.$ CRFIL RFD.$
IFFIL R.$ DLFIL R.$
IFFIL W.$ DLFIL W.$
IFFIL A.$ DLFIL A.$
IFFIL B.$ DLFIL B.$
IFFIL C.$ DLFIL C.$
IFFIL D.$ DLFIL D.$
IFFIL E.$ DLFIL E.$
IFFIL F.$ DLFIL F.$
...
```

Perform the backup or restore.

```
...
IFFIL DAE.$ DO BACKUPX D A E|DLFIL DAE.$
IFFIL DAF.$ DO BACKUPX D A F|DLFIL DAF.$
IFFIL DBE.$ DO BACKUPX D B E|DLFIL DBE.$
IFFIL DBF.$ DO BACKUPX D B F|DLFIL DBF.$
IFFIL DCE.$ DO BACKUPX D C E|DLFIL DCE.$
IFFIL DCF.$ DO BACKUPX D C F|DLFIL DCF.$
IFFIL DDE.$ DO BACKUPX D D E|DLFIL DDE.$
IFFIL DDF.$ DO BACKUPX D D F|DLFIL DDF.$
IFFIL WAE.$ DO BACKUPX W A E|DLFIL WAE.$
IFFIL WAF.$ DO BACKUPX W A F|DLFIL WAF.$
IFFIL WBE.$ DO BACKUPX W B E|DLFIL WBE.$
IFFIL WBF.$ DO BACKUPX W B F|DLFIL WBF.$
IFFIL WCE.$ DO BACKUPX W C E|DLFIL WCE.$
IFFIL WCF.$ DO BACKUPX W C F|DLFIL WCF.$
IFFIL WDE.$ DO BACKUPX W D E|DLFIL WDE.$
IFFIL WDF.$ DO BACKUPX W D F|DLFIL WDF.$
IFFIL REA.$ DO BACKUPX R E A|DLFIL REA.$
IFFIL REB.$ DO BACKUPX R E B|DLFIL REB.$
IFFIL REC.$ DO BACKUPX R E C|DLFIL REC.$
IFFIL RED.$ DO BACKUPX R E D|DLFIL RED.$
IFFIL RFA.$ DO BACKUPX R F A|DLFIL RFA.$
IFFIL RFB.$ DO BACKUPX R F B|DLFIL RFB.$
IFFIL RFC.$ DO BACKUPX R F C|DLFIL RFC.$
IFFIL RFD.$ DO BACKUPX R F D|DLFIL RFD.$
IFFIL M.$ DLFIL M.$
BEEP
```

Done!

Our subordinate DO-file, BACKUPX.DO, is as follows:

```
IFUSR 0{2}: BACKUPY {1} {2} {3} 0
IFUSR 1{2}: BACKUPY {1} {2} {3} 1
IFUSR 2{2}: BACKUPY {1} {2} {3} 2
...  (Repeat the command line for users 3 through 28)
IFUSR 29{2}: BACKUPY {1} {2} {3} 29
IFUSR 30{2}: BACKUPY {1} {2} {3} 30
```

```
IFUSR 31{2}: BACKUPY {1} {2} {3} 31
```

Our secondary subordinate DO-file, BACKUPY.DO is:

```
IFNOTCHR R {1} DELETE {4}{2}:*.BAK;N|DELETE {4}{2}:-PRINT-*.*;N
IFCHR W {1} SET {4}{2}:*.*;N-A
IFNOTCHR R {1} COPY {4}{2}: {4}{3}:;NAC
IFCHR R {1} SET {4}{3}:*.*;N-R
IFCHR R {1} IFFIL M.$ COPY {4}{2}: {4}{3}:;NX
IFCHR R {1} IFNOTFIL M.$ COPY {4}{2}: {4}{3}:;N
IFCHR R {1} IFCHR 0 {4} SET 0{3}:;NG
IFCHR R {1} SET {4}{3}:*.COM;NR
IFCHR R {1} SET {4}{3}:*.CMD;NR
```

Our sign-on message file, BACKUP.MSG, is:

```
------------------------------------------------------------------
^*                        SYSTEM BACKUP ROUTINE

            This routine will allow the BACKUP of data
          FROM any combination of drives A:, B:, C:, or D:
             TO either of the floppy drives E: or F:

                                OR

                        the RESTORATION of data
                     TO any drive A:, B:, C:, or D:
                  FROM either of the floppy drives E: or F:
------------------------------------------------------------------
```

Thus ends our complex example, which really shouldn't be so complex, taken a step at a time.

-- Source Codes --

Source codes for the entire family of conditional and conditional support utilities in both 8- and 16-bit versions are provided with this article. These source codes may be readily typed-in, assembled and linked, providing you with the power we have been discussing.

In order to save space, the 8- and 16-bit versions of the source codes are given in "parallel." A very cursory examination will show that the 8-bit code is given first, then the 16-bit code, then the common comments. A vertical bar "|" is used as a separator between the 8- and 16-bit codes.

Type in these codes in a normal manner, using only the 8-bit or only the 16-bit, whichever is proper for your system. (Please be certain to type in my copyright line, as I spent many long hours creating these utilities.) Assemble the finished code using Microsoft's Macro-80 (M80) for the 8-bit or Software 2000's TASM for the 16-bit. Use GEN.COM or TLINK.CMD to link the resultant modules into working command files.

Again to save space, common routines have been placed into four subroutine files. These files must be separately assembled, and are joined with the appropriate main modules at link time. The easiest way to prevent accidental

ommision of a subroutine module is to create the following one-line GEN files for the main modules involved. The main modules are always listed first:

```
IFFIL,SUBIF1,SUBIF2,SUBIF3      ;For IFFIL and IFNOTFIL

IFUSR,SUBIF1,SUBIF2             ;For IFUSR and IFNOTUSR

IFCHR,SUBIF1                    ;For IFCHR and IFNOTCHR

CRFIL,SUBIF2,SUBIF3,SUBIF4      ;For CRFIL and CRFILYES

DLFIL,SUBIF2,SUBIF3,SUBIF4      ;For DLFIL and DLFILYES

ENDDO,SUBIF4                    ;For ENDDO and ENDDOYES

DOHALT,SUBIF4                   ;For DOHALT

PROMPT,SUBIF2,SUBIF3,SUBIF4     ;For PROMPT

BEEP                            ;For BEEP

CLS                             ;For CLS
```

In order to create the IFNOTxxx and xxxxxYES files, create first the base file (IFFIL is the base file for IFNOTFIL, etc.) then use monitor (8-bit) or TBUG (16-bit) to create the variant. This may best be done via the following file, "IFPATCH.DO":

```
8-Bit                           16-Bit
-------------------------------------------------------
MONITOR                         TBUG {1}
L{1}.COM                        E0100
E0103                           FF
FF                              <cr>
<tab><cr>                       S{2}
S{2}.COM                        Q
Q
```

The "<tab><cr>" in the 8-bit DO-file will cause MONITOR to exit the Examine/change mode. The above DO-files are, of course, executed via the command <DO IFPATCH inputfile outputfile>, where "inputfile" would be "IFFIL", "IFUSR", etc., and "outputfile" would be "IFNOTFIL", "IFNOTUSR", etc.

For those who do not wish to spend the time typing, all the source codes, GENeration files, finished command files, and a complete copy of this article in both Wordstar and ready-to-print forms is available on an 8" single-sided/single-density CP/M format diskette directly from the author:

R. Roger Breton
836 Portola Av., Ste. 599
Livermove, CA 94550

Cost is $25.00 for TUG members, $35.00 for non members.

Source codes are also available at no charge via 1200 Baud modem. Modem

transfers are by appointment only, no exceptions. Call (415) 443-3131 Tuesday through Friday (no Monday calls, please) between 1:30 and 6:00 p.m. Pacific time for an appointment and protocols required.

Please bear in mind that these utilities are released to the public domain for personal use only, and may not legally be distributed, for sale or for free, without prior permission. Dealers wishing to provide these utilities to their customers may purchase a license for $75.00, which includes a master diskette of all source codes and documentation (also containing source and documentation for a few other goodies). Send a COMPANY check (as proof of doing business) to the above address.

-- Conclusion --

As may be seen by the above discussion and examples, TurboDOS' very powerful batch processing utilities, DO.COM and BATCH.COM provide considerable flexibility in the implementation of batch-processing functions. When coupled with the conditional utilities, IFFIL, IFNOTFIL, IFUSR, IFNOTUSR, IFCHR and IFNOTCHR, and the conditional-support utilities, CRFIL, CRFILYES, DLFIL, DLFILYES, ENDDO, ENDDOYES, DOHALT, PROMPT, BEEP, and CLS, the DO-files becomes practically a programming language. With a little imagination, extremely powerful DO-files can be created to better realize the potentials of a TurboDOS systems.

-- Combined 8- and 16-Bit Source Codes --

```
;Routine to generate the following conditional utilities:  IFFIL, IFNOTFIL
;
;Copyright (C) 1985, 1986, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   26 January 1986
;
        NAME ('IFFIL')          |       MODULE "IFFIL"          ;Program ID
        .Z80                    |                               ;Zilog mnemonics
                                |
                                |       LOC  Data#              ;Locate in data segment
                                |NOTFLG::BYTE 0                 ;NOT flag
                                |
        CSEG                    |       LOC  Code#              ;Locate in code segment
        JP   BEGIN              |       JMP  BEGIN              ;Skip
NOTFLG::DB   0                  |                               ;NOT flag
        DB   '-- Copyright (C) 1985, 1986, R. Roger Breton --'
                                |       BYTE "-- Copyright (C) 1985, 1986, "
                                |       BYTE "R. Roger Breton --"
                                |
BEGIN:  LD   A,(NOTFLG)         |BEGIN: CMP  BYTE NOTFLG,=0     ;Is NOT flag set?
        OR   A                  |
        JR   Z,NOINIT           |       JZ   __NNIT             ;If no, skip
        LD   A,-1               |       MOV  BYTE NOTFLG,=1     ;Initialize NOT flag
        LD   (NOTFLG),A         |
NOINIT: CALL TSTFIL##           |__NNIT: CALL TSTFIL#           ;Test for file specified
        OR   A                  |       OR   DX,DX              ;Error?
        JR   NZ,ABORT           |       JNZ  __ABT              ;If yes, abort
        CALL FNDFIL##           |       CALL FNDFIL#            ;Is file present?
        LD   B,A                |                               ;Move results
                                |       MOV  CL,=7              ;Make result TRUE/FALSE
                                |       SHR  AL,CL
        LD   A,(NOTFLG)         |       CMP  NOTFLG,AL          ;Process cmd list?
        XOR  B                  |
        JR   NZ,EXIT            |       JNZ  __EXIT             ;If no, done
        XOR  A                  |       XOR  AL,AL              ;Clear IFCHR flag
        CALL CMDLST##           |       CALL CMDLST#            ;Process command list
        OR   A                  |       OR   DX,DX              ;Error?
        JR   NZ,ABORT           |       JNZ  __ABT              ;If yes, abort
        LD   DE,0080H           |       MOV  DX,&0x0080         ;Point to command list
        LD   A,(0050H)          |                               ;Get version flag
        CP   0C3H               |                               ;Version 1.3x or 1.4x?
        JR   NZ,TDOS12          |                               ;If no, skip
        LD   C,18               |       MOV  CL,=18             ;Send command list
        CALL 0050H              |       INT  223
        JR   EXIT               |       JMP  __EXIT             ;Done
TDOS12: LD   C,108              |                               ;Send command list
        CALL 0005H              |
        JR   EXIT               |                               ;Skip
ABORT:  LD   C,9                |__ABT:  MOV  CL,=9             ;Print abort message
        CALL 0005H              |       INT  224
EXIT:   JP   0000H              |__EXIT: MOV  CL,=0             ;Return to o/s
                                |       INT  224
```

```
                                |
        END                     |       END

;Routine to generate the following conditional utilities:  IFUSR, IFNOTUSR
;
;Copyright (C) 1985, 1986, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   26 January 1986
;
        NAME ('IFUSR')          |       MODULE "IFUSR"          ;Program ID
        .Z80                    |                               ;Zilog mnemonics
                                |
        DSEG                    |       LOC  Data#              ;Locate in data segment
                                |NOTFLG::BYTE 0                 ;NOT flag
NDSMSG: DB   7,13,10,'No drive specified.',13,10,'$'
                                |NDSMSG: BYTE "\7\r\nNo drive specified.\r\n$"
NUSMSG: DB   7,13,10,'No user area specified.',13,10,'$'
                                |NUSMSG: BYTE "\7\r\nNo user area specified.\r\n$"
NPUMSG: DB   7,13,10,'Non-priviledged user.',13,10,'$'
                                |NPUMSG: BYTE "\7\r\nNon-priviledged user.\r\n$"
PRVFLG: DB   0                  |PRVFLG: BYTE 0                 ;Priviledged user flag
T14FLG::DB   0                  |T14FLG::BYTE 0                 ;Version 1.4x flag
SPCUSR::DB   80H                |SPCUSR::BYTE 0x80              ;Specified user number
CURUSR::DB   0                  |CURUSR::BYTE 0                 ;Current user number
DIRBUF: DB   128                |DIRBUF: RES  128               ;Directory DMA buffer
                                |
        CSEG                    |       LOC  Code#              ;Locate in code segment
        JP   BEGIN              |       JMP  BEGIN              ;Skip
NOTFLG::DB   0                  |                               ;NOT flag
        DB   '-- Copyright (C) 1985, 1986, R. Roger Breton --'
                                |       BYTE "-- Copyright (C) 1985, 1986, "
                                |       BYTE "R. Roger Breton --"
                                |
BEGIN:  LD   A,(NOTFLG)         |BEGIN: CMP  BYTE NOTFLG,=0     ;Is NOT flag set?
        OR   A                  |
        JR   Z,NOINIT           |       JZ   __NNIT             ;If no, skip
        LD   A,-1               |       MOV  BYTE NOTFLG,=1     ;Initialize NOT flag
        LD   (NOTFLG),A         |
NOINIT: LD   DE,NDSMSG          |__NNIT: MOV  DX,&NDSMSG        ;Point to no drive msg
        LD   A,(005CH)          |       CMP  BYTE 0x005C,=0     ;Is drive default?
        OR   A                  |
                                |       JNZ  __S1               ;If no, continue
        JP   Z,ABORT            |       JMP  __ABT              ;If yes, abort
        LD   A,(0050H)          |                               ;Version 1.22?
        CP   0C3H               |
        JP   NZ,SUSER           |                               ;If yes, skip
        LD   C,12               |__S1:   MOV  CL,=12            ;Get status/version
        CALL 0050H              |       INT  223
        LD   A,C                |       CMP  CL,=0x14           ;Version 1.4x
        CP   14H                |
        JP   NZ,SUSER           |       JNZ  __SUSR             ;If no, skip
        LD   A,-1               |       MOV  BYTE T14FLG,=1     ;Set version 1.4 flag
        LD   (T14FLG),A         |
        BIT  7,B                |       TEST CH,=0x80           ;Priviledged?
```

```
        JR   Z,NPRIV      |        JZ   __NPRV            ;If no, skip
        LD   A,-1         |        MOV  BYTE PRVFLG,=1    ;Set priviledged flag
        LD   (PRVFLG),A   |
NPRIV:  LD   DE,NUSMSG    |__NPRV: MOV  DX,&NUSMSG        ;Point to no user msg
        LD   A,(006BH)    |        CMP  BYTE 0x006B,=0    ;Was a user specified?
        OR   A            |
        JR   Z,ABORT      |        JZ   __ABT             ;If no, abort
        LD   E,-1         |        MOV  DL,=255           ;Get current user number
        LD   C,32         |        MOV  CL,=32
        CALL 0005H        |        INT  224
        LD   (CURSUR),A   |        MOV  CURUSR,AL         ;Stash it
        LD   B,A          |
        LD   A,(0069H)    |        MOV  AH,0x0069         ;Get spec user number
        CP   B            |        CMP  AL,AH             ;Same as current?
        JR   Z,SUSER      |        JZ   __SUSR            ;If yes, skip
        LD   (SPCUSR),A   |        MOV  SPCUSR,AH         ;Stash it
        LD   DE,NPUMSG    |        MOV  DX,&NPUMSG        ;Point to non-priv msg
        LD   A,(PRVFLG)   |        CMP  BYTE PRVFLG,=0    ;Priviledged?
        OR   A            |
        JR   Z,ABORT      |        JZ   __ABT             ;If no, abort
        CALL NEWUSR##     |        CALL NEWUSR#           ;Move to spec user area
SUSER:  LD   HL,005DH     |__SUSR: MOV  DI,&0x005D        ;Initialize FCB
        LD   (HL),'?'     |        MOV  AL,=63
        LD   D,H          |
        LD   E,L          |
        INC  DE           |
        LD   BC,11        |        MOV  CX,=11
        LDIR              |        REP STOS BYTE
        LD   DE,DIRBUF    |        MOV  DX,&DIRBUF        ;Set DMA buffer
        LD   C,26         |        MOV  CL,=26
        CALL 0005H        |        INT  224
        LD   DE,005CH     |        MOV  DX,&0x005C        ;Find any file
        LD   C,17         |        MOV  CL,=17
        CALL 0005H        |        INT  224
        PUSH AF           |        PUSH AX                ;Save result
        CALL OLDUSR##     |        CALL OLDUSR#           ;Return to current user
        POP  AF           |        POP  AX                ;Restore results
        LD   B,8          |        MOV  CL,=7             ;Make result TRUE/FALSE
TFLOOP: SRA  A            |        SHR  AL,CL
        DJNZ TFLOOP       |
        LD   B,A          |
        LD   A,(NOTFLG)   |        CMP  NOTFLG,AL         ;Process command list?
        XOR  B            |
        JR   NZ,EXIT      |        JNZ  __EXIT            ;If no, done
        XOR  A            |        XOR  AL,AL             ;Clear IFCHR flag
        CALL CMDLST##     |        CALL CMDLST#           ;Process command list
        OR   A            |        OR   DX,DX             ;Error?
        JR   NZ,ABORT     |        JNZ  __ABT             ;If yes, abort
        LD   DE,0080H     |        MOV  DX,&0x0080        ;Send command list
        LD   A,(0050H)    |                               ;Get T-function jump
        CP   0C3H         |                               ;Version 1.3x or 1.4x?
        JR   NZ,TDOS12    |                               ;If no, skip
        LD   C,18         |        MOV  CL,=18
        CALL 0050H        |        INT  223
        JR   EXIT         |        JMP  __EXIT            ;Done
```

```
ABORT:  LD   C,9          |__ABT:  MOV  CL,=9            ;Print abort message
        CALL 0005H        |        INT  224
EXIT:   JP   0000H        |__EXIT: MOV  CL,=0            ;Return to o/s
                          |        INT  224
                          |
        END               |        END

;Routine to generate the following conditional utilities:  IFCHR, IFNOTCHR
;
;Copyright (C) 1985, 1986, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   26 January 1986
;
        NAME ('IFCHR')    |        MODULE "IFCHR"        ;Program ID
        .Z80              |                              ;Zilog mnemonics
                          |
        DSEG              |        LOC  Data#            ;Locate in data segment
                          |NOTFLG::BYTE 0                ;NOT flag
NTCMSG: DB   7,13,10,'Illegal/missing test character.',13,10,'$'
                          |NTCMSG: BYTE "\7\r\nIllegal/missing test "
                          |        BYTE "character.\r\n$"
NMCMSG: DB   7,13,10,'Illegal/missing match character.',13,10,'$'
                          |NMCMSG: BYTE "\7\r\nIllegal/missing match "
                          |        BYTE "character.\r\n$"
                          |
        CSEG              |        LOC  Code#            ;Locate in code segment
        JP   BEGIN        |        JMP  BEGIN            ;Skip
NOTFLG::DB   0            |                              ;NOT flag
        DB   '-- Copyright (C) 1985, 1986, R. Roger Breton --'
                          |        BYTE "-- Copyright (C) 1985, 1986, "
                          |        BYTE "R. Roger Breton --"
                          |
BEGIN:  LD   A,(NOTFLG)   |BEGIN:  CMP  BYTE NOTFLG,=0   ;Is NOT flag set?
        OR   A            |
        JR   Z,NOINIT     |        JZ   __NNIT           ;If no, skip
        LD   A,-1         |        MOV  BYTE NOTFLG,=1   ;Initialize NOT flag
        LD   (NOTFLG),A   |
NOINIT: LD   DE,NTCMSG    |__NNIT: MOV  DX,&NTCMSG       ;Point to no tst chr msg
        LD   HL,005DH     |        CMP  BYTE 0x005D,=32  ;Was test chr specified?
        LD   A,' '        |
        CP   (HL)         |
        JR   Z,ABORT      |        JZ   __ABT            ;If no, abort
        LD   C,(HL)       |
        LD   DE,NMCMSG    |        MOV  DX,&NMCMSG       ;Point to no mch chr msg
        LD   HL,006DH     |        CMP  BYTE 0x006D,=32  ;Was mtch chr specified?
        CP   (HL)         |
        JR   Z,ABORT      |        JZ   __ABT            ;If no, abort
        LD   A,(HL)       |        MOV  AL,0x006D        ;Get match character
        SUB  C            |        SUB  AL,0x005D        ;Do the test
        JR   Z,GMATCH     |        JZ   __MCH            ;If a match, skip
        LD   A,-1         |        MOV  AL,1             ;Set for no match
GMATCH: LD   B,A          |
        LD   A,(NOTFLG)   |__MCH:  CMP  AL,NOTFLG        ;Process command list?
        XOR  B            |
```

```
        JR    NZ,EXIT     |        JNZ   __XIT             ;If no, skip
        LD    A,-1        |        MOV   AL,=1             ;Set IFCHR flag
        CALL  CMDLST##    |        CALL  CMDLST#           ;Process command list
        OR    A           |        OR    DX,DX             ;Error?
        JR    NZ,ABORT    |        JNZ   __ABT             ;If yes, abort
        LD    DE,0080H    |        MOV   DX,&0x0080        ;Send command list
        LD    A,(0050H)   |                                ;Get version flag
        CP    0C3H        |                                ;Version 1.3x or 1.4x?
        JR    NZ,TDOS12   |                                ;If no, skip
        LD    C,18        |        MOV   CL,=18
        CALL  0050H       |        INT   223
        JR    EXIT        |        JMP   __XIT             ;Skip
ABORT:  LD    C,9         |__ABT:  MOV   CL,=9             ;Print abort message
        CALL  0005H       |        INT   224
EXIT:   JP    0000H       |__XIT:  MOV   CL,=0             ;Return to o/s
                          |        INT   224
                          |
        END               |        END
```

```
;Routine to generate the following utilities:  CRFIL, CRFILYES
;
;Copyright (C) 1985, 1986, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   26 January 1986
;
        NAME ('CRFIL')  |        MODULE "CRFILE"         ;Program ID
        .Z80            |                               ;Zilog mnemonics
                        |
        DSEG            |        LOC  Data#             ;Locate in data segment
                        |YESFLG::BYTE 0                 ;YES flag
                        |CLSSTR::BYTE 26,"$$$$$$$$$$$$" ;Clear-screen string
FILMSG: DB    13,10,'File $'
                        |FILMSG: BYTE "\r\nFile $"
FLXMSG: DB    ' already exists.',7,13,10,'$'
                        |FLXMSG: BYTE " already exists.\7\r\n$"
UCRMSG: DB    ' unable to be created.',7,13,10,'$'
                        |UCRMSG: BYTE " unable to be created.\7\r\n$"
CRFMSG: DB    ' successully created.',13,10,'$'
                        |CRFMSG: BYTE " successully created.\r\n$"
PMTSPT::DB    -1        |PMTSPT::BYTE 1
PMTBEL::DB    -1        |PMTBEL::BYTE 1
PMTSTR::DB    13,10,'Okay to create $$$$$$$$$$$$$$$$'
PMTSTX::DB    ' at this time?  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$'
                        |PMTSTR::BYTE "\r\nOkay to create $$$$$$$$$$$$$$$$"
                        |PMTSTX::BYTE " at this time?  "
                        |        BYTE "$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$"
                        |
        CSEG            |        LOC  Code#             ;Locate in code segment
        JP    BEGIN     |        JMP  BEGIN             ;Skip
YESFLG::DB    0         |                               ;YES flag
CLSSTR::DB    26,'$$$$$$$$$$$$'                       ;Clear-screen string
        DB    '-- Copyright (C) 1985, 1986, R. Roger Breton --'
                        |        BYTE "-- Copyright (C) 1985, 1986, "
                        |        BYTE "R. Roger Breton --"
```

```
                        |
BEGIN:  CALL TSTFIL##   |BEGIN:  CALL TSTFIL#          ;Test for file specified
        OR   A          |        OR   DX,DX            ;Was it?
        JR   NZ,ABORT   |        JNZ  __ABT            ;If no, skip
        LD   A,(YESFLG) |        CMP  BYTE YESFLG,=0   ;Is YES flag set?
        OR   A          |
        JR   Z,NOYES    |        JZ   __NYES           ;If no, skip
        XOR  A          |        XOR  AL,AL            ;Clear long-prompt flag
        CALL CHKPMT##   |        CALL CHKPMT#          ;Chk and display prompt
        CALL GETCHR##   |        CALL GETCHR#          ;Get a reply character
        CP   'Y'        |        CMP  AL,=89           ;Is it a "Y"?
        JR   NZ,EXIT    |        JNZ  __EXIT           ;If no, exit to o/s
NOYES:  CALL FNDFIL##   |__NYES: CALL FNDFIL#          ;Check for file present
        LD   DE,FLXMSG  |        MOV  DX,&FLXMSG      ;Point to fl exists msg
        OR   A          |        OR   AL,AL            ;Did file exist?
        JR   Z,NOCRFL   |        JZ   __NOCR           ;If yes, abort
        CALL NEWUSR##   |        CALL NEWUSR#          ;Move to spec user
        LD   DE,005CH   |        MOV  DX,&0x005C       ;Create the file
        LD   C,22       |        MOV  CL,=22
        CALL 0005H      |        INT  224
        PUSH AF         |        PUSH AX               ;Save the error code
        LD   DE,005CH   |        MOV  DX,&0x005C       ;Close the file
        LD   C,16       |        MOV  CL,=16
        CALL 0005H      |        INT  224
        CALL OLDUSR##   |        CALL OLDUSR#          ;Return to original user
        POP  AF         |        POP  AX               ;Restore the error code
        LD   DE,UCRMSG  |        MOV  DX,&UCRMSG       ;Preset to no create msg
        OR   A          |        OR   AL,AL            ;Was file created?
        JR   NZ,ABORT   |        JNZ  __NOCR           ;If no, abort
        LD   DE,CRFMSG  |        MOV  DX,&CRFMSG       ;Point to created msg
NOCRFL: PUSH DE         |__NOCR: PUSH DX               ;Save message pointer
        LD   DE,FILMSG  |        MOV  DX,&FILMSG       ;Print output msg pt 1
        LD   C,9        |        MOV  CL,=9
        CALL 0005H      |        INT  224
        CALL PRTFIL##   |        CALL PRTFIL#          ;Print filename
        POP  DE         |        POP  DX               ;Restore message pointer
ABORT:  LD   C,9        |__ABT:  MOV  CL,=9            ;Display the message
        CALL 0005H      |        INT  224
EXIT:   JP   0000H      |__EXIT: MOV  CL,=0            ;Exit to o/s
                        |        INT  224
                        |
        END             |        END
```

```
;Routine to generate the following utilities:  DLFIL, DLFILYES
;
;Copyright (C) 1985, 1986, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   26 January 1986
;
        NAME ('DLFIL')  |        MODULE "DLFIL"        ;Program ID
        .Z80            |                              ;Zilog mnemonics
                        |
        DSEG            |        LOC  Data#            ;Locate in data segment
                        |YESFLG::BYTE 0                ;YES flag
```

```
                        |CLSSTR::BYTE 26,"$$$$$$$$$$$$" ;Clear-screen string
FILMSG: DB   13,10,'File $'
                        |FILMSG: BYTE "\r\nFile $"
NFLMSG: DB   ' does not exist.',7,13,10,'$'
                        |NFLMSG: BYTE " does not exist.\7\r\n$"
FFOMSG: DB   ' is a FIFO file, not deleted.',7,13,10,'$'
                        |FFOMSG: BYTE " is a FIFO file, not deleted.\7\r\n$"
ROFMSG: DB   ' is set READ ONLY.',7,13,10,'$'
                        |ROFMSG: BYTE " is set READ ONLY.\7\r\n$"
NMTMSG: DB   ' is not empty, not deleted.',7,13,10,'$'
                        |NMTMSG: BYTE " is not empty, not deleted.\7\r\n$"
NDLMSG: DB   ' unable to be deleted.',7,13,10,'$'
                        |NDLMSG: BYTE " unable to be deleted.\7\r\n$"
DFLMSG: DB   ' successully deleted.',13,10,'$'
                        |DFLMSG: BYTE " successully deleted.\r\n$"
PMTSPT::DB   -1         |PMTSPT::BYTE 1
PMTBEL::DB   -1         |PMTBEL::BYTE 1
PMTSTR::DB   13,10,'Okay to delete $$$$$$$$$$$$$$$$'
PMTSTX::DB   ' at this time?  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$'
                        |PMTSTR::BYTE "\r\nOkay to delete $$$$$$$$$$$$$$$$"
                        |PMTSTX::BYTE " at this time?  "
                        |       BYTE "$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$"
                        |
        CSEG            |       LOC  Code#              ;Locate in code segment
        JP   BEGIN      |       JMP  BEGIN              ;Skip
YESFLG::DB   0          |                               ;YES flag
CLSSTR::DB   26,'$$$$$$$$$$$$'                          ;Clear-screen string
        DB   '-- Copyright (C) 1985, 1986, R. Roger Breton --'
                        |       BYTE "-- Copyright (C) 1985, 1986, "
                        |       BYTE "R. Roger Breton --"
                        |
BEGIN:  CALL TSTFIL##   |BEGIN: CALL TSTFIL#            ;Test for file specified
        OR   A          |       OR   DX,DX              ;Was it?
        JR   NZ,ABORT   |       JNZ  __ABT              ;If no, skip
        LD   A,(YESFLG) |       CMP  BYTE YESFLG,=0     ;Is YES flag set?
        OR   A          |
        JR   Z,NOYES    |       JZ   __NYES             ;If no, skip
        XOR  A          |       XOR  AL,AL              ;Clear long-prompt flag
        CALL CHKPMT##   |       CALL CHKPMT#            ;Chk and display prompt
        CALL GETCHR##   |       CALL GETCHR#            ;Get a reply character
        CP   'Y'        |       CMP  AL,=89             ;Is it a "Y"?
        JR   NZ,EXIT    |       JNZ  __EXIT             ;If no, done
NOYES:  CALL FNDFIL##   |__NYES: CALL FNDFIL#           ;Is the file there?
        LD   DE,NFLMSG  |       MOV  DX,&NFLMSG        ;Point to no-file msg
        OR   A          |       OR   AL,AL              ;Was file found?
        JR   NZ,NODLFL  |       JNZ  __NODL             ;If no, abort
        LD   DE,NMTMSG  |       MOV  DX,&NMTMSG        ;Point to not-empty msg
        LD   HL,007DH   |       MOV  BX,&0x007D        ;Check file size
        LD   A,(HL)     |       MOV  AL,[BX]
        INC  HL         |
        OR   (HL)       |       OR   AL,1[BX]
        INC  HL         |
        OR   (HL)       |       OR   AL,2[BX]
        JR   NZ,NODLFL  |       JNZ  __NODL             ;If not zero, abort
        CALL NEWUSR##   |       CALL NEWUSR#            ;Move to proper user
```

```
        LD   DE,005CH   |        MOV  DX,&0x005C        ;Open the file
        LD   C,15       |        MOV  CL,=15
        CALL 0005H      |        INT  224
        LD   DE,005CH   |        MOV  DX,&0x005C        ;Close the file
        LD   C,16       |        MOV  CL,=16
        CALL 0005H      |        INT  224
        CALL OLDUSR##   |        CALL OLDUSR#           ;Back to current user
        LD   DE,FFOMSG  |        MOV  DX,&FFOMSG        ;Point to FIFO message
        LD   A,(005DH)  |        TEST BYTE 0x005D,=0x80 ;Is FIFO attribute set?
        BIT  7,A        |
        JR   NZ,NODLFL  |        JNZ  __NODL            ;If yes, abort
        LD   DE,ROFMSG  |        MOV  DX,&ROFMSG        ;Point to read-only msg
        LD   A,(0065H)  |        TEST BYTE 0x0065,=0x80 ;Is read-only attr set?
        BIT  7,A        |
        JR   NZ,NODLFL  |        JNZ  __NODL            ;If yes, abort
        CALL NEWUSR##   |        CALL NEWUSR#           ;Point to proper user
        LD   DE,005CH   |        MOV  DX,&0x005C        ;Delete the file
        LD   C,19       |        MOV  CL,=19
        CALL 0005H      |        INT  224
        PUSH AF         |        PUSH AX                ;Save the error code
        CALL OLDUSR##   |        CALL OLDUSR#           ;Back to current user
        POP  AF         |        POP  AX                ;Restore the error code
        LD   DE,NDLMSG  |        MOV  DX,&NDLMSG        ;Point to no-delete msg
        OR   A          |        OR   AL,AL             ;Good delete?
        JR   NZ,NODLFL  |        JNZ  __NODL            ;If no, abort
        LD   DE,DFLMSG  |        MOV  DX,&DFLMSG        ;Point to delete message
NODLFL: PUSH DE         |__NODL: PUSH DX                ;Save message pointer
        LD   DE,FILMSG  |        MOV  DX,&FILMSG        ;Print output msg pt 1
        LD   C,9        |        MOV  CL,=9
        CALL 0005H      |        INT  224
        CALL PRTFIL##   |        CALL PRTFIL#           ;Print filename
        POP  DE         |        POP  DX                ;Restore message pointer
ABORT:  LD   C,9        |__ABT:  MOV  CL,=9             ;Display the message
        CALL 0005H      |        INT  224
EXIT:   JP   0000H      |__EXIT: MOV  CL,=0             ;Exit to o/s
                        |        INT  224
                        |
        END             |        END

;Routine to generate the following utilities:  ENDDO, ENDDOYES
;
;Copyright (C) 1985, 1986, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   26 January 1986
;
        NAME ('ENDDO')  |        MODULE "ENDDO"         ;Program ID
        .Z80            |                               ;Zilog mnemonics
                        |
        DSEG            |        LOC  Data#             ;Locate in data segment
                        |YESFLG::BYTE 0                 ;YES flag
                        |CLSSTR::BYTE 26,"$$$$$$$$$$$$" ;Clear-screen string
PMTSPT::DB   0          |PMTSPT::BYTE 0
PMTBEL::DB   -1         |PMTBEL::BYTE 1
PMTSTR::DB   13,10,'Okay to abort the DO-file at this time? '
```

```
        DB    '$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$'
                          |PMTSTR::BYTE "\r\nOkay to abort the "
                          |        BYTE "DO-file at this time?  "
                          |        BYTE "$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$"
EDOMSG: DB    13,10,'DO-file aborted.',13,10,'$'
                          |EDOMSG: BYTE "\r\nDO-file aborted.\r\n$"
                          |
        CSEG              |        LOC  Code#              ;Locate in code segment
        JP    BEGIN       |        JMP  BEGIN              ;Skip
YESFLG::DB    0           |                                ;YES flag
CLSSTR::DB    26,'$$$$$$$$$$$'                           ;Clear-screen string
        DB    '-- Copyright (C) 1985, 1986, R. Roger Breton --'
                          |        BYTE "-- Copyright (C) 1985, 1986, "
                          |        BYTE "R. Roger Breton --"
                          |
BEGIN:  LD    A,(YESFLG)  |BEGIN:  CMP  BYTE YESFLG,=0      ;Is YES flag set?
        OR    A           |
        JR    Z,NOYES     |        JZ   __NYES             ;If no, skip
        XOR   A           |        XOR  AL,AL              ;Clear long-prompt flag
        CALL  CHKPMT##    |        CALL CHKPMT#            ;Chk and display prompt
        CALL  GETCHR##    |        CALL GETCHR#            ;Get a reply character
        CP    'Y'         |        CMP  AL,=89             ;Is it a "Y"?
        JR    NZ,EXIT     |        JNZ  __EXIT             ;If no, exit to o/s
NOYES:  LD    DE,0        |__NYES: MOV  DX,=0              ;Prepare to terminate
        LD    A,(0050H)   |                                ;Check the version
        CP    0C3H        |                                ;1.3x or 1.4x?
        JR    NZ,TDOS12   |                                ;If no, skip
        LD    C,16        |        MOV  CL,=16             ;Terminate the DO-file
        CALL  0050H       |        INT  223
        JR    EDDONE      |                                ;Skip
TDOS12: LD    C,98        |                                ;Terminate the DO-file
        CALL  0005H       |
EDDONE: LD    DE,EDOMSG   |        MOV  DX,&EDOMSG         ;Print terminated msg
        LD    C,9         |        MOV  CL,=9
        CALL  0005H       |        INT  224
EXIT:   JP    0000H       |__EXIT: MOV  CL,=0              ;Exit to o/s
                          |        INT  224
                          |
        END               |        END
```

```
;Routine to generate the following utility:  DOHALT
;
;Copyright (C) 1985, 1985, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   26 January 1986
;
        NAME ('DOHALT')   |        MODULE "DOHALT"         ;Program ID
        .Z80              |                                ;Zilog mnemonics
                          |
        DSEG              |        LOC  Data#              ;Locate in data segment
                          |        BYTE 0                  ;Dummy flag byte
                          |CLSSTR::BYTE 26,"$$$$$$$$$$$"   ;Clear-screen string
PMTSPT::DB    0           |PMTSPT::BYTE 0
PMTBEL::DB    -1          |PMTBEL::BYTE 1
```

```
PMTSTR::DB   13,10,'DO-file halted, press any key to continue.'
        DB   '  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$'
                        |PMTSTR::BYTE "\r\nDO-file halted, "
                        |        BYTE "press any key to continue."
                        |        BYTE "  $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$"
                        |
        CSEG            |        LOC  Code#              ;Locate in code segment
        JP   BEGIN      |        JMP  BEGIN              ;Skip
        DB   0          |                                ;Dummy flag byte
CLSSTR::DB   26,'$$$$$$$$$$$$'                          ;Clear-screen string
        DB   '-- Copyright (C) 1985, 1986, R. Roger Breton --'
                        |        BYTE "-- Copyright (C) 1985, 1986, "
                        |        BYTE "R. Roger Breton --"
                        |
BEGIN:  XOR  A          |BEGIN:  XOR  AL,AL              ;Clear long-prompt flag
        CALL CHKPMT##   |        CALL CHKPMT#            ;Chk and display prompt
        CALL GETCHR##   |        CALL GETCHR#            ;Get a reply character
        JP   0000H      |        MOV  CL,=0              ;Exit to o/s
                        |        INT  224
                        |
        END             |        END

;Routine to generate the following utility:  PROMPT
;
;Copyright (C) 1985, 1986, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   26 January 1986
;
        NAME ('PROMPT') |        MODULE "PROMPT"         ;Program ID
        .Z80            |                               ;Zilog mnemonics
                        |
        DSEG            |        LOC  Data#              ;Locate in data segment
                        |        BYTE 0                  ;Dummy flag byte
                        |CLSSTR::BYTE 26,"$$$$$$$$$$$$" ;Clear-screen string
FILMSG: DB   7,13,10,'File $'
                        |FILMSG: BYTE "\7\r\nFile $"
OSZMSG: DB   ' is too large.',13,10,'$'
                        |OSZMSG: BYTE " is too large.\r\n$"
FNFMSG: DB   ' not found.',13,10,'$'
                        |FNFMSG: BYTE " not found.\r\n$"
PMTSPT::DB   -1         |PMTSPT::BYTE 1
PMTBEL::DB   -1         |PMTBEL::BYTE 1
PMTSTR::DB   '$$$'      |PMTSTR::BYTE "$$$"
        DS   2048       |        RES  2048
                        |
        CSEG            |        LOC  Code#              ;Locate in code segment
        JP   BEGIN      |.       JMP  BEGIN              ;Skip
        DB   0          |                                ;Dummy flag byte
CLSSTR::DB   26,'$$$$$$$$$$$$'                          ;Clear-screen string
        DB   '-- Copyright (C) 1985, 1986, R. Roger Breton --'
                        |        BYTE "-- Copyright (C) 1985, 1986, "
                        |        BYTE "R. Roger Breton --"
                        |
BEGIN:  CALL TSTFIL##   |BEGIN:  CALL TSTFIL#            ;Test for file specified
```

```
        OR    A                 |         OR    DX,DX             ;Was it?
        JR    NZ,PRINT          |         JNZ   __PRT             ;If no, abort
        CALL  FNDFIL##          |         CALL  FNDFIL#           ;Find the prompt file
        LD    DE,FNFMSG         |         MOV   DX,&FNFMSG        ;Point to no file msg
        LD    A,B               |         OR    AL,AL             ;Was file found?
        OR    A                 |
        JR    NZ,ABORT          |         JNZ   __ABT             ;If no, abort
        LD    DE,OSZMSG         |         MOV   DX,&OSZMSG        ;Point to oversize msg
        LD    HL,007DH          |         MOV   BX,&0x007D        ;Check file size
        LD    A,(HL)            |         CMP   BYTE [BX],=17     ;Too many records?
        CP    17                |
        JR    NC,ABORT          |         JNC   __ABT             ;If yes, skip
        INC   HL                |         MOV   AL,1[BX]          ;Way too many records?
        LD    A,(HL)            |
        INC   HL                |
        OR    (HL)              |         OR    AL,2[BX]
        JR    NZ,ABORT          |         JNZ   __ABT             ;If yes, skip
        CALL  NEWUSR##          |         CALL  NEWUSR#           ;Move to specified user
        LD    DE,005CH          |         MOV   DX,&0x005C        ;Open the file
        LD    C,15              |         MOV   CL,=15
        CALL  0005H             |         INT   224
        LD    DE,PMTSTR         |         MOV   DX,&PMTSTR        ;Point to DMA buffer
        LD    B,16              |         MOV   CX,=16            ;Set count
LOOP1:  PUSH  BC                |__LP1:   PUSH  CX                ;Save count
        PUSH  DE                |         PUSH  DX                ;Save DMA address
        LD    C,26              |         MOV   CL,=26            ;Set it
        CALL  0005H             |         INT   224
        LD    DE,005CH          |         MOV   DX,&0x005C        ;Read a record
        LD    C,20              |         MOV   CL,=20
        CALL  0005H             |         INT   224
        POP   HL                |         POP   DX                ;Restore DMA address
        LD    DE,128            |         ADD   DX,=128           ;Add offset
        ADD   HL,DE             |
        EX    DE,HL             |
        POP   BC                |         POP   CX                ;Restore Count
        OR    A                 |         OR    AL,AL             ;Was it a good read?
        JR    NZ,LSTREC         |         JNZ   __LREC            ;If no, exit loop
        DJNZ  LOOP1             |         LOOP  __LP1             ;Do next record
LSTREC: LD    DE,005CH          |__LREC:  MOV   DX,&0x005C        ;Close the file
        LD    C,16              |         MOV   CL,=16
        CALL  0005H             |         INT   224
        CALL  OLDUSR##          |         CALL  OLDUSR#           ;Return to original user
        LD    A,-1              |         MOV   AL,=1             ;Set long-prompt flag
        CALL  CHKPMT##          |         CALL  CHKPMT#           ;Process file as string
        JR    EXIT              |         JMP   __EXIT            ;Skip
ABORT:  PUSH  DE                |__ABT:   PUSH  DX                ;Save message pointer
        LD    DE,FILMSG         |         MOV   DX,&FILMSG        ;Print file message
        LD    C,9               |         MOV   CL,=9
        CALL  0005H             |         INT   224
        CALL  PRTFIL##          |         CALL  PRTFIL#           ;Print filename
        POP   DE                |         POP   DX                ;Restore message pointer
PRINT:  LD    C,9               |__PRT:   MOV   CL,=9             ;Display the message
        CALL  0005H             |         INT   224
EXIT:   JP    0000H             |__EXIT:  MOV   CL,=0             ;Exit to o/s
                                |         INT   224
```

```
                                |
        END                     |       END

;Program to beep the console or the printer three times.
;
;Copyright (C) 1985, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   05 December 1985
;
        NAME ('BEEP')           |       MODULE "BEEP"           ;Program ID
        .Z80                    |                               ;Zilog mnemonics
                                |
        DSEG                    |       LOC  Data#              ;Locate in data segment
                                |PCLOCK::BYTE 8                 ;Processor clock rate
LSTFLG: DB   0                  |LSTFLG: BYTE 0                 ;List flag
                                |
        CSEG                    |       LOC  Code#              ;Locate in code segment
        JP   BEGIN              |                               ;Skip patch points
PCLOCK::DB   6                  |                               ;Processor clock rate
                                |
BEGIN:  CALL LCHECK             |       CALL LCHECK             ;Check for list option
        CALL RING               |       CALL RING               ;Ring the bell
        CALL WAIT               |       CALL WAIT               ;Wait
        CALL RING               |       CALL RING               ;Ring it again
        CALL WAIT               |       CALL WAIT               ;Wait
        CALL RING               |       CALL RING               ;Ring it one last time
        JP   0000H              |       MOV  CL,=0              ;Exit to o/s
                                |       INT  224
                                |
LCHECK: LD   HL,0080H           |LCHECK: MOV  BX,&0x0080        ;Point to command tail
        LD   A,(HL)             |                               ;Get 1st character
        CP   0                  |       CMP  BYTE [BX],=0       ;Is there a cmd tail?
        RET  Z                  |       JZ   __DONE             ;If no, done
LOOP1:  INC  HL                 |__LP1:  INC  BX                ;Point to next chr
        LD   A,(HL)             |                               ;Get it
        CP   0                  |       CMP  BYTE [BX],=0       ;End of command tail?
        RET  Z                  |       JZ   __DONE             ;If yes, done
        CP   ';'                |       CMP  BYTE [BX],=59      ;Semicolon?
        JR   NZ,LOOP1           |       JNZ  __LP1              ;If no, try again
LOOP2:  INC  HL                 |__LP2:  INC  BX                ;Point to next chr
        LD   A,(HL)             |                               ;Get it
        CP   0                  |       CMP  BYTE [BX],=0       ;End of command tail?
        RET  Z                  |       RZ   __DONE             ;If yes, done
        AND  5FH                |       AND  BYTE [BX],=0x5F    ;Make it upper-case
        CP   'L'                |       CMP  BYTE [BX],=76      ;Is it an "L"?
        JR   NZ,LOOP2           |       JNZ  __LP2              ;If no, try again
        LD   A,-1               |       MOV  BYTE LSTFLG,=1     ;Set the list flag
        LD   (LSTFLG),A         |
        RET                     |__DONE: RET                    ;Done
                                |
RING:   LD   E,7                |RING:   MOV  DL,=7             ;Get bell code
        LD   C,2                |        MOV  CL,=2             ;Preset for console
        LD   A,(LSTFLG)         |        CMP  BYTE LSTFLG,=0    ;Is list flag set?
        CP   0                  |
```

```
        JR   Z,RCON       |        JZ   __RCON          ;If no, skip
        LD   C,5          |        MOV  CL,=5           ;Set for list device
RCON:   CALL 0005H        |__RCON: INT  224             ;Send the bell
        RET               |        RET                  ;Done
                          |
WAIT:   LD   A,(PCLOCK)   |WAIT:   MOV  AL,PCLOCK       ;Set outer loop count
WTOLP:  LD   C,192        |__OLP:  MOV  DL,=194         ;Set middle loop count
WTMLP:  LD   B,121        |__MLP:  MOV  CX,=101         ;Set inner loop count
WTILP:  NOP               |__ILP:  NOP                  ;Waste a little time
        DJNZ WTILP        |        LOOP __ILP           ;Do inner loop
        DEC  C            |        DEC  DL              ;Dec middle loop counter
        JR   NZ,WTMLP     |        JNZ  __MLP           ;Do middle loop
        DEC  A            |        DEC  AL              ;Dec outer loop counter
        JR   NZ,WTOLP     |        JNZ  __OLP           ;Do outer loop
        RET               |        RET                  ;Done
                          |
        END               |        END
```

```
;Program to clear the console screen
;
;Copyright (C) 1985, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   03 December 1985
;
        NAME ('CLS')      |        MODULE "CLS"         ;Program ID
        .Z80              |                             ;Zilog mnemonics
                          |
                          |        LOC  Data#           ;Locate in data segment
                          |        BYTE 0               ;Dummy flag byte
                          |CLSSTR::BYTE 26,"$$$$$$$$$$$$"
                          |
        CSEG              |        LOC  Code#           ;Locate in code segment
        JP   BEGIN        |                             ;Skip
        DB   0            |                             ;Dummy flag byte
CLSSTR::DB   26,'$$$$$$$$$$$$'                          ;Clear-screen string
                          |
BEGIN:  LD   DE,CLSSTR    |        MOV  DX,&CLSSTR      ;Clear the screen
        LD      C,9       |        MOV  CL,=9
        CALL    0005H     |        INT  224
        JP      0000H     |        MOV  CL,=0           ;Exit to o/s
                          |        INT  224
                          |
        END               |        END
```

```
;If-Group subroutine:  CMDLST
;
;Copyright (C) 1985, 1986, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   26 January 1986
;
        NAME ('SUBIF1')   |        MODULE "SUBIF1"      ;Program ID
        .Z80              |                             ;Zilog mnemonics
                          |
```

```
        DSEG            |        LOC  Data#             ;Locate in data segment
MCLMSG: DB   7,13,10,'Missing command list.',13,10,'$'
                        |MCLMSG: BYTE "\7\r\nMissing command list.\r\n$"
                        |
        CSEG            |        LOC  Code#             ;Locate in code segment
CMDLST::LD   B,A        |                               ;Save the IFCHR flag
        LD   HL,0081H   |CMDLST::MOV  BX,&0x0081        ;Point to cmd tail space
        LD   C,' '      |                               ;Get a space
LOOP1A: LD   (HL),C     |__LP1A: MOV  BYTE [BX],=32     ;Space the character
        INC  HL         |        INC  BX                ;Point to the next one
        LD   A,(HL)     |                               ;Get it
        CP   0          |        CMP  BYTE [BX],=0      ;End of command tail?
        JR   Z,ENDCT    |        JZ   __NDCT            ;If yes, skip
        CP   9          |        CMP  BYTE [BX],=9      ;Tab?
        JR   Z,LOOP1A   |        JZ   __LP1A            ;If yes, space it/do nxt
        CP   C          |        CMP  BYTE [BX],=32     ;Space?
        JR   Z,LOOP1A   |        JZ   __LP1A            ;If yes, do next chr
LOOP1B: LD   (HL),C     |__LP1B: MOV  BYTE [BX],=32     ;Space the character
        INC  HL         |        INC  BX                ;Point to the next one
        LD   A,(HL)     |                               ;Get it
        CP   0          |        CMP  BYTE [BX],=0      ;End of command tail?
        JR   Z,ENDCT    |        JZ   __NDCT            ;If yes, skip
        CP   9          |        CMP  BYTE [BX],=9      ;Tab?
        JR   NZ,L1NTAB  |        JNZ  __NTAB            ;If no, skip
        LD   (HL),C     |        MOV  BYTE [BX],=32     ;Space it
L1NTAB: CP   C          |__NTAB: CMP  BYTE [BX],=32     ;Space?
        JR   NZ,LOOP1B  |        JNZ  __LP1B            ;If no, space it/do next
        LD   A,B        |                               ;Get IFCHR flag
        LD   B,0        |                               ;Clear it
        OR   A          |        OR   AL,AL             ;Was IFCHR flag set?
                        |        MOV  AL,=0            ;Clear it anyway
        JR   NZ,LOOP1A  |        JNZ  __LP1A            ;If yes, do it all again
LOOP2:  INC  HL         |__LP2:  INC  BX                ;Point to next character
        LD   A,(HL)     |                               ;Get it
LOOP3:  CP   0          |__LP3:  CMP  BYTE [BX],=0      ;End of command tail?
        JR   Z,ENDCT    |        JZ   __NDCT            ;If yes, skip
        CP   '|'        |        CMP  BYTE [BX],=124    ;Separator?
        JR   NZ,LOOP2   |        JNZ  __LP2             ;If no, do next chr
        INC  HL         |        CMP  BYTE 01[BX],=124  ;Next chr a separator?
        LD   A,(HL)     |
        DEC  HL         |
        CP   '|'        |
        JR   Z,SEPCHR   |        JZ   __SEP             ;If yes, skip
        LD   (HL),'\'   |        MOV  BYTE [BX],=92     ;Convert separator
        JR   LOOP2      |        JMP  __LP2             ;Do next character
SEPCHR: LD   (HL),' '   |__SEP:  MOV  BYTE [BX],=32     ;Replace with a space
LOOP4:  INC  HL         |__LP4:  INC  BX                ;Do next character
        LD   A,(HL)     |
        CP   '|'        |        CMP  BYTE [BX],=124    ;Separator?
        JR   Z,LOOP4    |        JZ   __LP4             ;If yes, bypass it
        JR   LOOP3      |        JMP  __LP3             ;Back to the main loop
ENDCT:  LD   (HL),0     |__NDCT: MOV  BYTE [BX],=0      ;Mark end of cmd tail
        LD   A,L        |                               ;Get position
        SUB  129        |        SUB  BX,=0x0081        ;Get length
        LD   (0080H),A  |        MOV  0x0080,BL         ;Set length
```

```
        LD   HL,0080H     |        MOV  BX,&0x0080        ;Point to command tail
LOOP5:  INC  HL           |__LP5:  INC  BX                ;Point to next character
        LD   A,(HL)       |                               ;Get it
        CP   0            |        CMP  BYTE [BX],=0      ;End of command list?
        JR   Z,NOLIST     |        JZ   __NLST            ;If yes, skip
        CP   ' '          |        CMP  BYTE [BX],=32     ;Other than a space?
        JR   NZ,LDSEP     |        JNZ  __LDSP            ;If yes, skip
        JR   LOOP5        |        JMP  __LP5             ;Check next character
NOLIST: LD   DE,MCLMSG    |__NLST: MOV  DX,&MCLMSG       ;Point to missing msg
        LD   A,-1         |                               ;Set error return flag
        RET               |        RET                    ;Done
LDSEP:  CP   '\'          |__LDSP: CMP  BYTE [BX],=92     ;Leading separator?
        JR   NZ,DONE      |        JNZ  __DONE            ;If no, done
        LD   (HL),' '     |        MOV  BYTE [BX],=32     ;Stuff a space
        LD   HL,0081H     |        MOV  BX,=0x0081        ;Point to 1st list chr
        LD   (HL),A       |        MOV  BYTE [BX],=92     ;Stuff leading separator
DONE:   XOR  A            |__DONE: XOR  DX,DX             ;Clear error return flag
        RET               |        RET                    ;Done
                          |
        END               |        END

;If-Group subroutines:  NEWUSR, OLDUSR
;
;Copyright (C) 1985, 1986, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   26 January 1986
;
        NAME ('SUBIF2')   |        MODULE "SUBIF2"        ;Program ID
        .Z80              |                               ;Zilog mnemonics
                          |
        CSEG              |        LOC  Code#             ;Locate in code segment
NEWUSR::LD   A,(T14FLG##)
                          |NEWUSR::CMP  BYTE T14FLG#,=1   ;Version 1.4x?
        OR   A            |
        RET  Z            |        JNZ  __RET             ;If no, done
        LD   A,(SPCUSR##)
                          |        TEST BYTE SPCUSR#,=0x80;Was user specified?
        BIT  7,A          |
        RET  NZ           |        JNZ  __RET             ;If no, done
        LD   E,A          |        MOV  DL,SPCUSR#        ;Move to specified user
        LD   C,32         |        MOV  CL,=32
        CALL 0005H        |        INT  224
        RET               |__RET:  RET                    ;Done
                          |
OLDUSR::LD   A,(T14FLG##)
                          |OLDUSR::CMP  BYTE T14FLG#,=1   ;Version 1.4x?
        OR   A            |
        RET  NZ           |        JNZ  __RET             ;If no, done
        LD   A,(SPCUSR##)
                          |        TEST BYTE SPCUSR#,=0x80;Was user prev changed?
        BIT  7,A          |
        RET  NZ           |        JNZ  __RET             ;If no, done
        LD   A,(CURUSR##)
                          |        MOV  DL,CURUSR#        ;Move to original user
```

```
        LD   E,A        |
        LD   C,32       |        MOV  CL,=32
        CALL 0005H      |        INT  224
        RET             |__RET:  RET                      ;Done
                        |
        END             |        END

;If-Group subroutines:  TSTFIL, FNDFIL, PRTFIL
;
;Copyright (C) 1985, 1986, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   26 January 1986
;
        NAME ('SUBIF3') |        MODULE "SUBIF3"          ;Program ID
        .Z80            |                                ;Zilog mnemonics
                        |
        DSEG            |        LOC  Data#              ;Locate in data segment
NFSMSG: DB   7,13,10,'No filename specified.',13,10,'$'
                        |NFSMSG: BYTE "\7\r\nNo filename specified.\r\n$"
AFLMSG: DB   7,13,10,'Ambiguous filename specified.',13,10,'$'
                        |AFLMSG: BYTE "\7\r\nAmbiguous filename specified.\r\n$"
NPUMSG: DB   7,13,10,'Non-priviledged user.',13,10,'$'
                        |NPUMSG: BYTE "\7\r\nNon-priviledged user.\r\n$"
PRVFLG: DB   0          |PRVFLG: BYTE 0                  ;Priviledged user flag
T14FLG::DB   0          |T14FLG::BYTE 0                  ;Version 1.4x flag
SPCUSR::DB   80H        |SPCUSR::BYTE 0x80               ;Specified user area
CURUSR::DB   0          |CURUSR::BYTE 0                  ;Current user area
                        |
        CSEG            |        LOC  Code#              ;Locate in code segment
TSTFIL::LD   DE,NFSMSG  |TSTFIL::MOV  DX,&NFSMSG         ;Point to no file msg
        LD   HL,005DH   |        MOV  BX,&0x005D         ;Point to 1st fn chr
        LD   A,(HL)     |                               ;Get it
        CP   ' '        |        CMP  BYTE [BX],=32      ;Was filename specified?
        RET  Z          |        JZ   __RET              ;If no, done
        LD   DE,AFLMSG  |        MOV  DX,&AFLMSG         ;Point to ambiguous msg
        LD   A,'?'      |                               ;Get a question mark
        LD   B,11       |        MOV  CX,=11             ;Set count
TFLOOP: CP   (HL)       |__LP:   CMP  BYTE [BX],=63      ;Is fn/ft ambiguous?
        RET  Z          |        JZ   __RET              ;If yes, done
        INC  HL         |        INC  BX                 ;Point to next character
        DJNZ TFLOOP     |        LOOP __LP               ;Do 'em all
        LD   A,(0050H)  |                               ;Get T-function jump
        SUB  0C3H       |                               ;Version 1.30 or later?
        RET  NZ         |                               ;If no, done
        LD   C,12       |        MOV  CL,=12             ;Get version number
        CALL 0050H      |        INT  223
        BIT  7,B        |        TEST CH,=0x80           ;Is user priviledged?
        JR   Z,TFSKIP   |        JZ   __SKP              ;If no, skip
        LD   A,-1       |        MOV  BYTE PRVFLG,=1     ;Set priviledged flag
        LD   (PRVFLG),A |
TFSKIP: LD   A,C        |__SKP:  CMP  CL,=0x14           ;Is this version 1.4x?
        CP   14H        |
        RET  NZ         |        JNZ  __DONE             ;If no, done
        LD   A,-1       |        MOV  BYTE T14FLG,=1     ;Set version 1.4x flag
```

```
        LD   (T14FLG),A |
        LD   HL,006BH   |        CMP  BYTE 0x006B,=0    ;Was user specified?
        LD   A,(HL)     |
        OR   A          |
        RET  Z          |        JZ   __DONE            ;If no, done
        DEC  HL         |        MOV  AL,0x0069         ;Get user area
        DEC  HL         |
        LD   A,(HL)     |
        LD   (SPCUSR),A |        MOV  SPCUSR,AL         ;Save it
        LD   E,-1       |        MOV  DL,=255           ;Get current user
        LD   C,32       |        MOV  CL,=32
        CALL 0005H      |        INT  224
        LD   (CURUSR),A |        MOV  CURUSR,AL         ;Save it
        LD   B,A        |        SUB  AL,SPCUSR         ;Same as specified user?
        LD   A,(SPCUSR) |
        SUB  B          |
        RET  Z          |        JZ   __DONE            ;If yes, done
        LD   DE,NPUMSG  |        MOV  DX,&NPUMSG        ;Point to non-priv msg
        LD   A,(PRVFLG) |        CMP  BYTE PRVFLG,=1    ;Priviledged?
        SUB  -1         |
                        |__DONE: XOR  DX,DX             ;Clear error msg pointer
        RET             |__RET:  RET                   ;Done
                        |
FNDFIL::CALL NEWUSR##   |FNDFIL::CALL NEWUSR#           ;Move to specified user
        LD   DE,005CH   |        MOV  DX,&0x005C        ;Check for file present
        LD   C,35       |        MOV  CL,=35
        CALL 0005H      |        INT  224
        PUSH AF         |        PUSH AX                ;Save error code
        CALL OLDUSR##   |        CALL OLDUSR#           ;Move to original user
        POP  AF         |        POP  AX                ;Restore error code
        RET             |        RET                    ;Done
                        |
PRTFIL::LD   A,(SPCUSR) |PRTFIL::TEST BYTE SPCUSR,=0x80 ;Was user specified?
        BIT  7,A        |
        JR   NZ,PFNUSR  |        JNZ  __NUSR            ;If no, skip
                        |        MOV  AL,SPCUSR         ;Get user area code
        LD   E,0        |        MOV  DL,=0             ;Preset for 0-9
        CP   10         |        CMP  AL,=10            ;User area less than 10?
        JR   C,PFPRTU   |        JC   __PRTU            ;If yes, skip
        LD   E,'1'      |        MOV  DL,=49            ;Preset for 10-19
        SUB  10         |        SUB  AL,=10            ;Subtract 10
        CP   10         |        CMP  AL,=10            ;Less than 20?
        JR   C,PFPRTU   |        JC   __PRTU            ;If yes, skip
        INC  E          |        INC  DL                ;Preset for 20-29
        SUB  10         |        SUB  AL,=10            ;Subtract 10
        CP   10         |        CMP  AL,=10            ;Less than 30?
        JR   C,PFPRTU   |        JC   __PRTU            ;If yes, skip
        INC  E          |        INC  DL                ;Preset for 30 or 31
        SUB  10         |        SUB  AL,=10            ;Subtract 10
PFPRTU: PUSH AF         |__PRTU: PUSH AX                ;Save the value
        XOR  A          |        OR   DL,DL             ;User 0-9?
        CP   E          |
        JR   Z,PFPRT2   |        JZ   __PRT2            ;If yes, skip
        CALL PFPCHR     |        CALL __PCHR            ;Print first digit
PFPRT2: POP  AF         |__PRT2: POP  AX                ;Restore value
```

```
        ADD   A,48          |        ADD  AL,=48            ;Make it ASCII
        LD    E,A           |        MOV  DL,AL             ;Print second digit
        CALL  PFPCHR        |        CALL __PCHR
PFNUSR: LD    HL,005CH      |__NUSR: CMP  BYTE 0x005C,=0    ;Default drive?
        LD    A,(HL)        |
        OR    A             |
        JR    Z,PFNDRV      |        JZ   __NDRV            ;If yes, skip
                            |        MOV  DL,0x005C         ;Get the drive code
        ADD   A,64          |        ADD  DL,=64            ;Make it ASCII
        CALL  PFPCHR        |        CALL __PCHR            ;Print it
        JR    PFACOL        |        JMP  __ACOL            ;Skip
PFNDRV: LD    A,(SPCUSR)    |__NDRV: TEST BYTE SPCUSR,=0x80 ;Was user specified?
        BIT   7,A           |
        JR    NZ,PFNDRU     |        JNZ  __NDRU            ;If no, skip
PFACOL: LD    E,':'         |__ACOL: MOV  DL,=58            ;Print a colon
        CALL  PFPCHR        |        CALL __PCHR
PFNDRU: LD    HL,005CH      |__NDRU: MOV  BX,&0x005C        ;Point to the FCB
        LD    B,8           |        MOV  CX,=8             ;Set the count
        LD    A,' '         |                               ;Set the compare byte
        CALL  PFFNFT        |        CALL __PFNT            ;Print the filename
        LD    E,'.'         |        MOV  DL,=46            ;Print a period
        CALL  PFPCHR        |        CALL __PCHR
        LD    B,3           |        MOV  CX,=3             ;Set count
        CALL  PFFNFT        |        CALL __PFNT            ;Print the filetype
        RET                 |        RET                    ;Done
PFFNFT: INC   HL            |__PFNT: INC  BX                ;Point to next character
        LD    E,(HL)        |        MOV  DL,[BX]           ;Get it
        RES   7,E           |        AND  DL,=0x7F          ;Clear any attributes
        CP    E             |        CMP  DL,=32            ;Is it a space?
        JR    Z,PFNTSP      |        JZ   __NTSP            ;If yes, skip
        CALL  PFPCHR        |        CALL __PCHR            ;Print it
PFNTSP: DJNZ  PFFNFT        |__NTSP: LOOP __PFNT            ;Repeat for all ft chrs
        RET                 |        RET                    ;Done
PFPCHR: PUSH  HL            |__PCHR: PUSH BX                ;Save pointer
        PUSH  BC            |        PUSH CX                ;Save counter
        PUSH  AF            |                               ;Save compare byte
        LD    C,2           |        MOV  CL,=2             ;Print the character
        CALL  0005H         |        INT  224
        POP   AF            |                               ;Restore compare byte
        POP   BC            |        POP  CX                ;Restore counter
        POP   HL            |        POP  BX                ;Restore pointer
        RET                 |        RET                    ;Done
                            |
        END                 |        END

;If-Group subroutines:  CHKPMT, GETCHR
;
;Copyright (C) 1985, 1986, R. Roger Breton
;Author:  R. Roger Breton
;Version: 2.20
;Dated:   26 January 1986
;
        NAME ('SUBIF4')     |        MODULE "SUBIF4"        ;Program ID
        .Z80                |                               ;Zilog mnemonics
                            |
```

```
        DSEG            |        LOC  Data#             ;Locate in data segment
ECOMSG: DB   ' ',13,10,'$'
                        |ECOMSG: BYTE " \r\n$"
                        |LPTFLG: BYTE 0                ;Long-prompt flag
                        |
        CSEG            |        LOC  Code#             ;Locate in code segment
CHKPMT::PUSH AF         |CHKPMT::MOV  LPTFLG,AL         ;Save long-prompt flag
        LD   HL,PMTSTR##|        MOV  BX,&PMTSTR#       ;Point to prompt string
        OR   A          |        OR   AL,AL             ;Long or short prompt?
        JR   NZ,CPLPMT  |        JNZ  __LPMT            ;If long, skip
        LD   HL,0080H   |        MOV  BX,&0x0080        ;Point to command tail
        LD   A,(HL)     |                               ;Get length
        CP   0          |        CMP  BYTE [BX],=0      ;Is length zero?
        JR   Z,CPNPMT   |        JZ   __NPMT            ;If so, no promptstring
CPLUP1: INC  HL         |__LP1:   INC  BX               ;Point to next character
        LD   A,(HL)     |                               ;Get it
        CP   0          |        CMP  BYTE [BX],=0      ;End of command tail?
        JR   Z,CPNPMT   |        JZ   __NPMT            ;If so, no promptstring
        CP   ';'        |        CMP  BYTE [BX],=59     ;Is it a semicolon?
        JR   NZ,CPLUP1  |        JNZ  __LP1             ;If no, keep looking
        INC  HL         |        INC  BX                ;Get next character
CPLPMT: LD   A,(HL)     |__LPMT: CMP  BYTE [BX],=94     ;1st chr a circumflex?
        CP   '^'        |
        JR   NZ,CPASTR  |        JNZ  __ASTR            ;If no, skip
        CALL CPPCLS     |        CALL __CLS             ;Clear the screen
        CP   '*'        |        CMP  BYTE [BX],=42     ;Is 2nd chr an asterisk?
        JR   NZ,CPPSTR  |        JNZ  __PSTR            ;If no, do promptstring
        CALL CPNBEL     |        CALL __NBEL            ;Turn off bell
        JR   CPCSTR     |        JMP  __PSTR            ;Do promptstring
CPASTR: CP   '*'        |__ASTR: CMP  BYTE [BX],=42     ;Is 1st chr an asterisk?
        JR   NZ,CPPSTR  |        JNZ  __PSTR            ;If no, do promptstring
        CALL CPNBEL     |        CALL __NBEL            ;Turn off bell
        CP   '^'        |        CMP  BYTE [BX],=94     ;2nd chr a circumflex?
        JR   NZ,CPPSTR  |        JNZ  __PSTR            ;If no, do promptstring
        CALL CPPCLS     |        CALL __CLS             ;Clear the screen
CPPSTR: CP   0          |__PSTR: CMP  BYTE [BX],=0      ;End of prompt?
        JR   Z,CPNPMT   |        JZ   __NPMT            ;If so, no promptstring
                        |        MOV  SI,BX            ;Set source to pointer
        LD   BC,2048    |        MOV  CX,=2048          ;Preset long-prompt cnt
        POP  AF         |                               ;Restore PROMPT flag
        OR   A          |        CMP  BYTE LPTFLG,=0    ;Long or short prompt?
        JR   NZ,CPENDL  |        JNZ  __ENDL            ;If long, skip
        LD   BC,76      |        MOV  CX,=76            ;Set short-prompt count
        PUSH BC         |        PUSH CX                ;Save count
        LD   DE,PMTSTR##+02
                        |        MOV  DI,&PMTSTR#+2    ;Set dest to pmtstr
        LDIR            |        REP MOVS BYTE          ;Transfer promptstring
        POP  BC         |        POP  CX                ;Restore count
        LD   A,0        |        MOV  AL,=0             ;Get a null
        LD   HL,PMTSTR##|        MOV  DI,&PMTSTR#       ;Set scan to pmtstr
        CPIR            |        REPNZ SCAS BYTE        ;Find the end
        OR   B          |        OR   CX,CX             ;End of count?
        JR   Z,CPENDP   |        JZ   __ENDP            ;If yes, skip
        DEC  HL         |        DEC  DI                ;Point to the end
CPENDP: LD   (HL),' '   |__ENDP: MOV  BYTE [DI],=32     ;End promptstring there
```

```
        INC   HL          |
        LD    (HL),' '    |        MOV  BYTE 1[DI],=32
        INC   HL          |
        LD    (HL),'$'    |        MOV  BYTE 2[DI],=36
CPENDL: LD    A,0         |__ENDL: MOV  BYTE PMTSPT#,=0   ;Clear split-prompt flag
        LD    (PMTSPT##),A
CPNPMT: LD    A,(PMTBEL##)
                          |__NPMT: CMP  BYTE PMTBEL#,=0   ;Is bell flag set?
        OR    A           |
        JR    Z,CPSBEL    |        JZ   __SBEL            ;If no, skip
        LD    E,7         |        MOV  DL,=7             ;Beep the console
        LD    C,2         |        MOV  CL,=2
        CALL  0005H       |        INT  224
CPSBEL: LD    DE,PMTSTR## |__SBEL: MOV  DX,&PMTSTR#       ;Point to promptstring
        LD    A,(PMTSPT##)
                          |        CMP  BYTE PMTSPT#,=0   ;Split-prompt flag set?
        OR    A           |
        JR    Z,CPPPMT    |        JZ   __PPMT            ;If no, skip
        LD    C,9         |        MOV  CL,=9             ;Print promptstring
        CALL  0005H       |        INT  224
        CALL  PRTFIL##    |        CALL PRTFIL#           ;Print the filename
        LD    DE,PMTSTX## |        MOV  DX,&PMTSTX#      ;Point to rest of pstr
CPPPMT: LD    C,9         |__PPMT: MOV  CL,=9             ;Print it
        CALL  0005H       |        INT  224
        RET               |        RET                   ;Done
CPPCLS: PUSH  HL          |__CLS:  PUSH BX                ;Save the pointer
        LD    DE,CLSSTR## |        MOV  DX,&CLSSTR#      ;Clear the screen
        LD    C,9         |        MOV  CL,=9
        CALL  0005H       |        INT  224
        POP   HL          |        POP  BX                ;Restore the pointer
        JR    CPSSKP      |        JMP  __SSKP            ;Skip
CPNBEL: LD    A,0         |__NBEL: MOV  BYTE PMTBEL#,=0   ;Turn off bell
        LD    (PMTBEL##),A
CPSSKP: LD    (HL),13     |__SSKP: MOV  BYTE [BX],=13     ;Overwrite with a CR
        INC   HL          |        INC  BX                ;Point to next character
        LD    A,(HL)      |                               ;Get it
        RET               |        RET                   ;Done
                          |
GETCHR::LD    E,-1        |GETCHR::MOV  DL,=255           ;Look for an input chr
        LD    C,6         |        MOV  CL,=6
        CALL  0005H       |        INT  224
        OR    A           |        OR   AL,AL             ;Was a chr waiting?
        JR    Z,GETCHR    |        JZ   GETCHR            ;If no, look again
        AND   5FH         |        AND  AL,=0x5F          ;Make upper-case ASCII
        CP    'Y'         |        CMP  AL,=89            ;Is it a "Y"?
        JR    Z,ECHOY     |        JZ   __ECOY            ;If yes, skip
        LD    A,'N'       |        MOV  AL,=78            ;Get an "N"
ECHOY:  LD    (ECOMSG),A  |__ECOY: MOV  ECOMSG,AL        ;Save it
        LD    DE,ECOMSG   |        MOV  DX,&ECOMSG       ;Echo it
        LD    C,9         |        MOV  CL,=9
        CALL  0005H       |        INT  224
        LD    A,(ECOMSG)  |        MOV  AL,ECOMSG        ;Restore the character
        RET               |        RET                   ;Done
                          |
        END               |        END
```

ADJUSTING OLDER DRIVERS TO VERSION 1.43

R. Roger Breton and John E. Lauber

As those of you upgrading from earlier versions know, the 16-bit drivers for version 1.43 require some changes. The area of primary concern is the method used to implement a poll routine. In the older versions (prior to 1.43), a three-word semaphore was required in the data segment and a two-word link was required in the code segment immediately ahead of the poll routine, as show in this sample:

```
        LOC     Data#                   ;Locate in data segment
;
POLSPH: WORD    0x0000                  ;Event semaphore
__PSPH: WORD    __PSPH
        WORD    __PSPH
;
        LOC     Code#                   ;Locate in code Segment
;
        MOV     DX,&POLLNK              ;Get linkage address
        CALL    LNKPOL#                 ;Actvate the poll routine
;
        CALL    POLRTN                  ;Optional pre-test
        MOV     BX,&POLSPH              ;Get semaphore address
        CALL    WAIT#                   ;Wait for the event
;
        ...
;
POLLNK: WORD    0x0000                  ;Poll routine linkage
        WORD    0x0000
POLRTN: IN      AL,=STAT                ;Get device status
        TEST    AL,=MASK                ;Did event occur?
        JZ      POLXIT                  ;If no, done
;
        MOV     BX,&POLSPH              ;Get semaphore address
        CALL    SIGNAL#                 ;Signal the event
;
        MOV     BX,&POLLNK              ;Get linkage address
        CALL    UNLINK#                 ;Deactivate poll routine
;
POLXIT: RET                             ;Done
```

In the 1.43-and-later environment, the poll linkage must be moved to the data segment, with a pointer to the poll routine appended:

```
        LOC     Data#                   ;Locate in data segment
;
POLSPH: WORD    0x0000                  ;Event semaphore
__PSPH: WORD    __PSPH
        WORD    __PSPH
POLLNK: WORD    0x0000                  ;Poll routine linkage
        WORD    0x0000
        WORD    &POLRTN
;
        LOC     Code#                   ;Locate in code Segment
```

```
;
        MOV     DX,&POLLNK              ;Get linkage address
        CALL    LNKPOL#                 ;Actvate the poll routine
;
        CALL    POLRTN                  ;Optional pre-test
        MOV     BX,&POLSPH              ;Get semaphore address
        CALL    WAIT#                   ;Wait for the event
;
        ...
;
POLRTN: IN      AL,=STAT                ;Get device status
        TEST    AL,=MASK                ;Did event occur?
        JZ      POLXIT                  ;If no, done
;
        MOV     BX,&POLSPH              ;Get semaphore address
        CALL    SIGNAL#                 ;Signal the event
;
        MOV     BX,&POLLNK              ;Get linkage address
        CALL    UNLINK#                 ;Deactivate poll routine
;
POLXIT: RET                             ;Done
```

This whole arrangement would normally require that two sets of drivers be kept, the older style and the 1.43-and-later style. John Lauber has developed a method of creating version-independent drivers that is both simple and elegant. His method operates around a byte that is set to 00 in earlier versions and to FF in version 1.43+, thus providing an on-the-fly method of controlling routing. His code for so doing may be found in the following extract from an actual driver, and consists of the subroutines LNKPLC and UNLNKC and a "different" method of specifying the poll linkages:

```
        LOC     Data#                   ; locate in data segment
;
;parallel port Semaphore
;
POTSPH: WORD    0                       ; semaphore count
__PARL: WORD    __PARL
        WORD    __PARL
;
        LOC     Code#                   ; locate in code segment
;
PAROUT::MOV     DX,=PORTB               ; get status port
        IN      AX,DX                   ; check status
        AND     AH,=1                   ; is it ready?
        MOV     AH,CL                   ; get output char
        JNZ     __PAROT                 ; poll if busy
;
        MOV     DX,=PORTA               ; get data port
        OUT     DX,AX                   ; send data byte
        RET
;
__PAROT:MOV     POCHAR,AH               ; save the output char.
        MOV     DX,&PARPL               ; point to poll routine
        CALL    LNKPLC                  ; link it on
;
```

```
        MOV     BX,&POTSPH              ; point to semaphore
        JMP     WAIT#                   ; wait till output ready
;
        LOC     Data#                   ; locate in data segment
PARPL:  WORD    0
        WORD    0                       ; poll linkages
        WORD    &PARPR                  ; poll routine entry
        RELOC                           ; locate back in code segment
        WORD    0
        WORD    0                       ; poll linkages
;
PARPR:  MOV     DX,=PORTB               ; get status port
        IN      AX,DX                   ; check status
        AND     AH,=1                   ; is it ready?
        JNZ     __PXIT                  ; exit if not
;
        MOV     AH,POCHAR               ; get output char
        MOV     DX,=PORTA               ; data port address
        OUT     DX,AX                   ; send it
;
        MOV     BX,&PARPL               ; remove from poll list
        CALL    UNLNKC
        MOV     BX,&POTSPH              ; signal as ready
        JMP     SIGNAL#
;
__PXIT: RET                             ; return results
;
; Link poll routine common.
; Checks for TurboDOS poll version global and determines proper way
; to link a poll routine.
; On entry: DX => poll linkage structure in Data segment.
;
LNKPLC: MOV     AL,GEV143#              ; load version global
        TEST    AL,AL                   ; greater than or equal v1.43?
        JNZ     __1                     ; if so, continue
        MOV     BX,DX                   ; else, move pointer to reg
        MOV     DX,4[BX]                ; load code pointer
        SUB     DX,=4                   ; adjust for linkages
__1:    JMP     LNKPOL#                 ; and continue routine
;
; Un-link poll routine common.
; Checks for TurboDOS poll version global and determines proper way
; to un-link a poll routine.
;
UNLNKC: MOV     AL,GEV143#              ; load version global
        TEST    AL,AL                   ; greater than or equal v1.43?
        JNZ     __1                     ; if so, continue
        MOV     BX,4[BX]                ; else, load code pointer
        SUB     BX,=4                   ; adjust for linkages
__1:    JMP     UNLINK#                 ; and continue routine
```

The whole method pivots around the byte GEV143##. If this byte is 00, then the driver presumes 1.42 or earlier and acts accordingly. If this byte is not 00, then the driver presumes for 1.43 or later. The easiest method to set this byte is to add the module GEV143.0 to the GENeration files of all 1.43 systems. In

this manner, if the module is left out, TurboDOS will link the system and set the label "GEV143" equal to "UndData", returning a 00 to the LNKPLC and UNLNKC subroutines. If the module is included, the the label "GEV143" will be hard-coded to FF. The source code for the GEV143 module is:

```
#TITLE  "TURBODOS OPERATING SYSTEM VERSION DEPENDENT VARIABLE"
#PAGE   132,60
;
; Author:  John E. Lauber
;
        MODULE  "GEV143"
;
; Greater than or equal to version 1.43 patchable variable.
;
GEV143::BYTE    0XFF                    ; Default to >= version 1.43
;
        END
```

Thank you, John.