CHAPTER 4:
BATCH FILE TECHNIQUES

Introduction

Batch files are very useful in many different phases of a system's configuration and on-going operation. Therefore, this chapter will review some batch file programming methods before there is any further discussion of system setup.

As with PC-DOS, MOS's command processor is responsible for executing batch files. In addition to the DOS capabilities, the PC-MOS operating system provides a number of enhanced batch language features. These include the ability for one batch file to call another, full control of color and cursor position in the creation of display screens, built-in support for menus and other interactive prompts, and the ability to perform unique operations based on the task number.

Designing For Speed and Clarity

The most significant factor affecting the execution speed of a batch file is the disk response time. Presuming you are using a hard disk, the "RAMdisk effect" provided by the caching subsystem is usually quite adequate for small to moderately sized files. If you will be making significant use of batch files, especially large ones or ones which rely heavily on branching, refer to Chapter 5 for information on installing the $RAMDISK.SYS device driver. You can have your AUTOEXEC.BAT copy often used batch files up to the RAMdisk at bootup time and include the RAMdisk in the PATH statement.

The design of branching is another major influence on a batch file's performance. Branching is accomplished in MOS through the use of the GOTO and CALL commands. GOTO, as in PC-DOS, involves a jump to a label without an implicit return. The CALL statement, in a manner similar to many programming languages, involves a branch where an automatic branch back to the point of call will be made when a RETURN statement is encountered.

Each time a label associated with a branch must be found, the batch file is searched from its beginning to the line containing the label. Therefore, it is important to organize the batch file with performance tuning factors in mind. When a fast response is desired to a particular branch, or when a branch will be repeated often, it is best to begin a batch file as shown in listing 4-1. Following the first line are the batch commands which are treated as sub- procedures or goto points. When more than one branch is involved, organize the file so that the most frequently used portions are nearest the beginning.

Any branch that will be used infrequently should be put at the end of the file. One example of this is the group of batch statements which are used to handle errors. Also note that the REM statements which document this batch process should not be placed at the beginning of the file. The distance of often used labels from the start of the file should be kept as small as possible for best performance.

Listing 4-1

An example of batch file construction:

NOTE: comments enclosed in { }'s are not actually part of the batch file.

```
goto start              {skip around the procedure}

:procedure1             {this label will be found quickly since}
...                     {it is near the beginning of the file}
...
return

:start
....
call procedure1
...
if errorlevel 1 goto error1
call procedure1
abort

:error1                 {place infrequently used labels at the end}
....
abort
rem this is where the documentation goes
```

An additional performance tuning tip is to use the ABORT or STOP command when it is necessary to exit from a batch process from some point other than the end of the file. Using one of these commands will cause an immediate exit with no scan of the entire file as would be required if a "goto end" were used. The choice of STOP or ABORT depends on how MOS's nesting feature applies to your file. If your batch file will never be called from another batch file then either command will work.

If you need to terminate the batch file A2.BAT, when it will be called in a nested fashion from A1.BAT, then you must consider the conditions involved in the termination. If the STOP command is used to terminate batch file A2 then batch file A1 will continue processing after the nest call. However, if the conditions involved require a total exit from batch processing, regardless of any nesting that may be in effect, the ABORT command should be used. Note that when a return is made from a nested call an environment variable could be used to pass an error status indication back to the caller. Environment variable usage will be discussed in more detail later in this chapter.

When an error condition is detected from within a batch sub-procedure which requires an immediate termination of the batch process, STOP or ABORT may be used. There is no need to return from the sub-procedure first as both of these commands will erase the information that was stored to handle the RETURN function. Likewise, a GOTO statement may be used to branch out of a batch file sub-procedure to a common error handler before terminating the job (e.g. to display an error message). However, this should be the only case where an abnormal exit is taken from a sub-procedure.

By making nested calls to individual batch files, rather than using a set of sub-procedures within one main batch file, another degree of optimization may be possible. If the size of each sub-procedure is fairly small, placing the statements within the main batch file just after the goto start statement would be most efficient.

In small to moderate sized batch files, the re-reading of the file to locate a label will usually not involve a disk access since the data required will be found within the disk cache. If the sub-processes will involve a significant number of batch statements or involve the use of text screens, placing these sub-processes in separate batch files and making nested calls to them will often provide the best performance.

Splitting a batch process up into several files does increase the efforts involved in maintenance however. When changes are required, the editing of multiple files may be involved. For this reason, you may find it easier to develop and test your process as one large batch file before splitting it up into a family of files.

When making up the names for a set of related batch files, using the same letters for the first 3 or 4 characters of each file name will provide a good means of classifying these files as being related. For example, a multiscreen menu program which is used to activate commonly used applications programs might be named APMENU.BAT. Using APMSCR1.BAT for the statements required to paint the first screen and APMSCR2.BAT for the second, etc., means that entering DIR APM*.BAT will let you review all files in the set. Likewise, COPY APM*.BAT \XYZDIR can be used to copy these files.

Through careful design and documentation, it is possible to set up a library of standard batch file sub-procedures which can be called from a variety of batch processes. Maintaining a log of what each library routine does and which batch files use it is highly recommended. Using one file (e.g. BATLIB.DOC) to document all such library files will centralize this effort.

When external commands such as system utilities and applications programs are invoked from within a batch file, prefacing the command's name with the explicit directory path in which it is located will also improve performance. Including the command's path, and drive letter when necessary, eliminates the search of all directories listed in the PATH statement otherwise done by the command processor. If you later move any programs accessed through this method to a different directory, or rename their current directory, you must remember to update all batch files which make such explicit references. Using the SEARCH command on all batch files can simplify this. For example:

```
SEARCH /I "OLDPATH" *.BAT
```

Using Command Line Parameters

As in DOS, the first nine command line parameters are immediately available to a batch file through the use of the predefined system variables %1 through %9. The %0 parameter is also supported and, when more than nine parameters must be accessed, the NEXT command may be used to shift any additional ones down into the %1 through %9 set. MOS's NEXT command is equivalent in function to DOS's SHIFT command. See the section in this chapter on Compatibility for a trick to allow batch files to be oblivious to this naming difference. This will allow a batch file to run under either operating system and make use of multiple parameters.

When designing a batch file which relies on certain command line parameters being supplied it becomes important to be able to test for the existence of these parameters. This is done by testing the corresponding variables for a null value.

For example, if your batch file requires that a file name be specified for the %1 variable, the following statement will verify that a parameter was given:

```
if (%1) == () goto no_parm_error
```

In the case where the use of a command line parameter is optional, a sequence of tests like the following may be used:

```
if (%1) == () goto noparm
if %1 == a goto pointa
if %1 == A goto pointa
if %1 == b goto pointb
...
```

Note that once the first statement has been passed it is no longer necessary to include any extra characters with the %1 variable since it is known to have a non-null value. If the first statement were not used as a pretest, when the statement "if %1 == a goto pointa" was processed where the value of %1 was null, the command processor would see the malformed statement "if == a goto pointa" and produce a syntax error message. Note that if your situation can always guarantee a proper value for %1, you could simply start such a batch file with the statement GOTO POINT%1.

Through MOS's capability to prompt the user from within a batch file, a batch process may be designed to ask for any parameters which are not supplied on the command line. This may be accomplished through the use of the KEY statement to prompt for a single character or a yes/no type of response or to determine which selection a user requests from menu.

With a utility program such as MAKESET, discussed later in this chapter, a prompt may be made for a word or an entire string. MAKESET could even be used to cause a batch process to read in parameters from a configuration file.

When a nested call is made from one batch file to another, any command line parameters placed after the name of the batch file being called will be assigned to the %1 through %9 batch variables in that secondary batch process. Upon return to the calling batch file, the values of the %1 through %9 variables will be restored to their original value before the nested call. To pass information from a nested batch file back to the calling one, environment variables are the best choice.

The batch file XDEL.BAT, shown in listing 4-2, is an example of how the null parameter test technique can be used in conjunction with the NEXT statement and a loop structure to produce a batch file which can operate with a variable number of command line parameters. This batch file is useful in doing hard disk housekeeping since it makes it simple to erase multiple files with a single command line. To use XDEL, enter one or more file specifications after the batch file's name:

```
XDEL *.BAK TEMP1 FILE2.OLD
```

Listing 4-2, XDEL.BAT

NOTE: comments enclosed in { }'s are not actually part of the batch file.

```
:loop
if (%1) == ( ) abort        {when %1 is null, all parameters processed}
echo deleting %1
del %1                      {do the deletion}
if errorlevel 1 goto delerror
next                        {2nd parameter --> %1, 3rd --> %2, etc}
goto loop                   {go back to :loop and check for more}
:delerror
echo error in delete command
```

Using a loop structure in this way can be very useful. Also see the example of ALLBUT.BAT the EXCEPT and ONLY section of Chapter 11. The batch file COPYTO.BAT, as shown in listing 4-3, is a variant of this loop technique where the first parameter is used in a different manner than all subsequent parameters. This batch file regards the first parameter as the destination drive and/or path to which a group of files is to be copied. Following are one or more source file specifications, in the same format as would be used for a copy command. For example, to copy the files FILE1.TXT, STATUS.REP and all files with the extension .DOC to the A: drive, enter the following:

```
COPYTO A: FILE1.TXT STATUS.REP *.DOC
```

Listing 4-3, COPYTO.BAT

NOTE: comments enclosed in { }'s are not actually part of the batch file.

```
if (%2) == ( ) goto parmerr          {two parameters required}
set destination=%1                   {initial %1 value saved in environment}
if errorlevel 1 goto enverr          {get out if environment too small}
:loop
next                                 {2nd parameter --> %1, 3rd --> %2, etc}
if (%1) == ( ) goto done             {when %1 is null, all parameters processed}
echo copying %1
copy %1 %destination%                {do the copy}
if errorlevel 1 goto copyerr
goto loop                            {go back to :loop and check for more}
:done
set destination=                     {cleanup}
abort
:parmerr
echo at least two parameters must be given
echo.
echo proper form: copyto destination source1 source2 ...
abort
:enverr
echo aborting copyto.bat due to environment error
abort
:copyerr
echo aborting copyto.bat due to copy error
```

The %A Parameter

In addition to the %0 through %9 system variables, MOS supports a %A variable which can be used to determine the current task number from within a batch file. Upon finding the %A variable within a batch file statement, the command processor will determine the number of the current task and replace the %A with that number.

For consistency, two digits are always used in this replacement. For example, the statement IF NOT %A == 00 GOTO SETUPBG will translate to IF NOT 01 == 00 GOTO SETUPBG when run within task 1.

In Chapter 6 it will be shown how one AUTOEXEC.BAT file could be used as the startup batch file for tasks other than the foreground through the use of the %A variable. By testing the value of %A with an IF statement, unique action may be taken depending on the task number. In the simplest case, the flow of control may be directed one way for task 0 and another way for all other tasks. When an explicit action is required for each task, a set of if statements must be used to test for each case. If you can be sure that the task number will always be a value that you've allowed for, using %A to form the label for a GOTO statement can be the simplest and fastest approach. This is shown in listing 4-4.

Listing 4-4

Using %a in the formation of a goto:

```
...
...             {statements common to all tasks}
goto task%a
:task00
...             {statements unique to task 0}
...
abort
:task01
...             {statements unique to task 01}
...
abort
...
```

Another important use of %A is in the formation of file or path names. Sometimes the processing of a batch file will result in the creation of one or more output files. If nothing is done to insure that the filenames used will be unique for each task, conflicts can arise when more than one user attempts to access such a batch file at the same time. Not allowing simultaneous access to a batch file is an inconvenient and unnecessary restriction, as well as a tedious one to enforce.

Consider a batch file which invokes the DIR command and re-directs its display output into the file DIRLOG using the statement DIR DIRLOG. The use of this batch file by more than one task at the same time and within the same directory will result in the output from multiple invocations of DIR being mixed together in one DIRLOG file. By specifying the filename as DIRLOG%A instead, multiple accesses will be coordinated properly. When run in the foreground, DIRLOG00 will result with task 1 producing DIRLOG01, etc. Note that no more than six characters may be used in the base portion of a filename which includes the %A variable and no more than one additional character may be used with %A in the formation of a file's extension.

When working with certain types of software packages you may find it necessary to license and install an individual copy of the program for each user. This can be simplified for the user by creating one common batch file which uses the %A variable to form part of the path which prefaces the command name. For example, to individualize the application XYZAPP.EXE in this manner, you could install the foreground's copy in a directory named \XYZ\TASK00, with task #1's copy in \XYZ\TASK01, etc.

The user's of the system could then be supplied with just one simple batch file which would automate the selection of the proper copy of this program based on the task. This batch file would contain the line C:\XYZ\TASK%A\XYZAPP %1 %2 %3. (The command line parameters allow the applications parameters to be specified after the batch file's name.)

MOS's %A variable may also be used to provide isolation when one copy of an application program is being shared among a number of users. This variable can be used in the formation of a path name which uniquely identifies the location of each user's data files. This could be done by including %A within part of the parameter line for an application which is started for all users from within one common batch file.

Using Environment Variables

Through the use of the SET command, strings of text may be placed into a storage area called the environment. This storage area is managed by the command processor and is unique to each task. At any given time, only one master environment exists within a task even though there may be more than one secondary copy of the environment in existence. The master environment is the one which is the direct recipient of strings entered with the SET command and secondary copies are duplicates of this string buffer which are provided for each resident utility or transient application used within a task.

Within a batch file, an environment string may be referenced through the use of the environment variable's name. In the example of COPYTO.BAT, earlier in this chapter, the initial value of the %1 command line parameter was set into the environment variable DESTINATION and then later referenced by surrounding the variable name with percent signs: %DESTINATION%. When the command processor encounters a string enclosed within percent signs it will search the environment for a matching variable and substitute the associated string in place of the variable name before any further processing of the batch file line is done.

In the event that no match is found, an environment variable will be replaced with a null value. This must be considered whenever using such variables in a batch file where it is presumed that the variable will already exist. Also, when a variable is SET directly within the batch file within which it will be referenced, you must guard against an unexpected null value which could result from making an assignment from a command line parameter (%1 through %9) or from trying to set a string into the environment when there is no more room. The former case is addressed through the use of a null testing IF statement, and the latter through the inclusion of an IF ERRORLEVEL test after each SET statement or a liberal sizing of the environment.

Just as with %1-%9 and %A, environment variables may be used in the formation of a label for a GOTO or CALL statement. Another interesting application is to place an environment variable on a line by itself so that its replacement string will be processed as a command.

A simple example of this is illustrated in listing 4-5. More elaborate use of this method will be covered later in this section in the discussion of a context sensitive help feature for batch file menus.

Listing 4-5

Using an environment variable for a command:

```
...
...
set abc=echo this command came from the environment
...
%abc%
```

When using environment variables within a batch file it is important to always clean up upon exit by deleting any temporary variables. This measure of housekeeping is necessary to insure that there will always be enough room available and to reduce clutter. Be sure to cover all methods by which the batch job can be terminated.

When working with nested batch files where the called batch file determines that an error condition exists which warrants a complete termination of the batch process (e.g. through ABORT), special considerations must be made to insure that environment variables are deleted, especially any that are only used within the calling batch file.

One solution to this situation is to establish the convention that the called batch file provide an indication to the calling batch file when an error occurs. To accomplish this batch file to batch file message passing, another environment variable will be used, for example ERRCODE.

Consider the case where batch file A makes a nested call to batch file B and B detects an error. Batch file B will do a SET ERRCODE=1 statement and make an immediate return to the calling batch file (A) through the use of the STOP statement. When no error is detected in B, the ERRCODE variable will be SET to 0 before returning. Upon return from the nested call batch file A will perform a test of the value of ERRCODE as follows:

```
IF NOT %ERRCODE% == 0 GOTO CLEANUP
```

After the label :CLEANUP in batch file A will be a set of statements which perform any necessary cleanup operations before reporting the error condition to the user and terminating the batch job.

As mentioned previously in this chapter, when using environment variables in your system (regardless of whether they serve as temporary variables within a batch file) as a batch file to batch file message medium or as global attributes which are set once in the task's startup batch file, it is very important to maintain documentation on their usage. It is also vitally important the all users maintain this system log properly and coordinate any modifications in the use of environment variables and similar system resources.

Say, for example, a task's startup batch file creates a certain environment variable called CONFIG. Say, also, that the person who installed the system intends the value of CONFIG to remain unchanged throughout the operation of a task, since it will be tested by certain other batch files. These batch files will reference CONFIG in order to ascertain details about the machine's configuration.

If someone later creates a new batch file and the name CONFIG seems appropriate for an internal environment variable which they have need for, the original value of CONFIG will be overwritten. If you are going to use environment variables, the effort involved in creating and diligently maintaining a documentation file will help prevent the frustrating effort involved in debugging.

Sizing the Environment

The point at which concern must be given to the maximum possible size for the environment area is when something is done to place a ceiling on its growth. This capping can occur when a TSR type of program is loaded -- a program that stays resident in memory. As long as no TSR's are loaded and the SET statement is being invoked directly from the command line or from within a batch file, the limit on the environment's maximum size is approximately 55K. However, when a program has installed itself in memory in a resident manner, the current size of the environment is the maximum possible. In addition, during the processing of a nested batch file, a temporary ceiling is imposed on the environment in a similar fashion to a TSR style program.

It is important, therefore, to pre-plan by estimating the minimum acceptable environment size for each task. The ENVSIZE command is used to reserve space in memory for the environment. It is typically included in a task's startup batch file, before any TSR programs are installed. In general, optimum memory usage will be achieved when events are arranged as in listing 4-6. In this example, ENVSIZE is executed first, followed by any SET statements required as a preface to loading TSR utilities. Then comes the actual installation of TSR utility programs, followed by any additional SET statements.

Listing 4-6

An example of loading order in a startup batch file:

```
...
envsize 400
...
set xyzdata=c:\xyz\notes
c:\xyz\xyzpopup
...
set config=b1
set dbmspath=c:\dbdata
...
```

The reason for this particular loading order is that a copy of the current environment is allocated from a task's TPA when a TSR type of program is installed. This copy is solely for the use of the TSR, just in case it has been designed to expect that certain environment strings have been set. By entering only those environment strings necessary before loading TSR's, the remaining TPA memory can be kept as large as possible.

Special consideration must be given to the case where a task's startup batch file makes a nested call to another batch file which is also sometimes used as a startup batch file itself. This is discussed in Chapter 6 in the section on startup batch files. Also note that when the ADDDEV command is used to load a device driver in a task specific manner that the net effect is the same as loading a TSR program, with the same limiting of environment growth. Refer to Chapter 11 for more details on ADDDEV.

The AUTOCD Command

Sometimes it is necessary in the design of a batch file to be able to change the default drive and/or directory and then be able to restore the original default values before terminating. MOS's AUTOCD command can be used, in conjunction with the CD command, to provide this capability. This can be very useful with applications that are designed so that they must be run from within the directory in which their files exist. This is usually required in order for the application to access their associated data files and program overlays.

When the CD command is invoked without a command line parameter the current default setting will be displayed. By using the ">" symbol to re-direct this display output to a file, a record of the current default settings can be saved. Once this is done, the current drive and directory may be changed as many times as required. When the batch operation is complete the AUTOCD command can be used to read the contents of the file to which CD's output was sent and use that information to re-instate the original default values.

An important consideration when using this technique is that the file which receives the re-directed output of the CD command must be specified through the use of an explicit path. This applies to both the CD and AUTOCD statements. In addition, since the creation of a temporary file is involved, attention must be paid to deleting this file during the batch process's cleanup phase and documenting the use of this file to help avoid conflicts. Finally, when the possibility of simultaneous execution by more than one user exists, the %A system variable should be used in the formation of this temporary file name to make it task specific.

Listing 4-7

INSXYZ.BAT - use of AUTOCD in the installation of a TSR:

NOTE: comments enclosed in { }'s are not actually part of the batch file.

```
cd > c:\insxyz%a        {save a record of the current drive/path.}
c:
cd \xyz                 {make c:\xyz the current drive\path to}
xyzpopup                {accommodate xyzpopup's limitations.}
autocd c:\insxyz%a      {restore the original drive/path}
del c:\insxyz%a         {and clean up the temporary file.}
```

Listing 4-7 illustrates how AUTOCD could be used in the batch file INSXYZ.BAT to install a TSR which can only find its data files in the current directory. By locating this batch file in a directory which is included in the command search path, it could be invoked at any time no matter what the current drive and directory.

Prompts and Menus

Under PC-DOS the primary method of displaying text to the screen from within a batch file is through the ECHO command. In MOS, while ECHO is certainly supported, an extended type of screen painting capa- bility has been provided through the TEXT command. TEXT works with blocks of display information rather than with single lines. Following the TEXT keyword, all subsequent lines in the batch file are displayed until the ENDTEXT keyword is encountered:

```
TEXT

        This line will be displayed
        And so will this one

ENDTEXT
```

This command is especially useful in the construction of menus, prompts and informative text screens. Video attributes such as inverse video or specific foreground and background colors may be specified for the entire screen or for any portion of it. It is also possible to control the position of the cursor. Refer to the PC-MOS USER GUIDE for a complete list of these control codes.

When designing a TEXT screen which will make use of display attributes, it is easiest to use a full screen editor to first lay out and review the text without any control codes included. Once you have put together the wording and have determined the position of each portion of the display, then go back and add the attribute codes. This will make judging the spacing of the text on your screen much easier.

When you are ready to review TEXT screens that you've designed in this manner, first place them in a batch file by themselves, without any of the other batch file logic. When more than one screen is involved, you could put them all into one batch file separated by pause statements. Once you have completely reviewed and fine tuned each display to your satisfaction, it's time to add the remaining batch file statements.

An alternative approach you may wish to consider would be to keep each screen in its own batch file and make nested calls to them from the main batch process. This is an especially efficient approach when the screens you are building will be used by more than one batch file.

MOS's extended batch command set also includes a KEY statement which allows a batch process to prompt the user for a character and generate a response based on this character. As always, the quickest way to understand a new feature is to study an example, such as the following:

```
KEY %%A IN (A-D) DO GOTO POINT%%A
```

In a similar manner as the FOR statement, KEY works with a variable of the type %%A, %%B, etc. When this command is encountered within a batch file, batch processing is suspended until the user presses a key within the set of acceptable characters. This set is defined within the key command, in between the parentheses. There is no distinction between upper and lower case with this command.

When a key is pressed which is within the defined set, the batch statement to the right of the "DO" keyword is processed as if it were entered as a command by itself. Any occurrences of the %%variable will have been translated into the character that was entered. Unlike the FOR command, once a key has been pressed and the right hand portion of the KEY statement has been processed as a command there is no implicit return to the KEY statement for another iteration.

The most direct way to use this command is to include the %%variable in the formation of a label for a CALL or GOTO statement. If a KEY statement is preceded by a TEXT block, a simple menu could be implemented as shown in listing 4-8. The KEY command can also be used to form a simple yes/no type of prompt as illustrated in listing 4-9.

The set of characters which can be specified with KEY are A-Z, 0-9, and F1-F10 (for functions keys F1, F2, etc.), CR for the carriage return key and SP for the space bar. Also, explicit entries and ranges may be mixed if desired, such as: (A-D,F1,SP).

Listing 4-8

A simple menu:

```
goto start
:pointa
. . .
return
:pointb
. . .
return
:pointc
. . .
return
:pointd
abort
:start
cls
text

                A. Word Processing

                B. Data Base

                C. Spreadsheet

                D. Exit this menu

        Enter a letter (A-D)

endtext
key %%a in (a-d) do call point%%a
goto start
```

Listing 4-9

A yes/no prompt:

```
echo  Are you sure?  (Y/N)
key %%a in (y,n) do goto choice%%a
:choicey
...
:choicen
...
```

When constructing menus and prompts, you may find it useful in some situations to have more than one label at the same point in the batch file. For instance, if you wish to present a yes/no type of prompt where pressing the enter key will respond the same as if a "Y" were keyed, you could arrange the commands as in listing 4-10. Also, if you will be building multiple menus or prompts which will re-use the same letter choices you must use different base portions to comprise unique labels:

```
...
KEY %%A IN (A-C) DO GOTO MENU1%%A
...
...
KEY %%A IN (A-D) DO GOTO MENU2%%A
```

Listing 4-10

A yes/no prompt which includes the enter key:

```
echo  Are you sure?  (y/n)          (or press enter for yes)
key %%a in (y,n,cr) do goto choice%%a
:choicey
:choicecr
...
:choicen
...
```

Through the combined use of several of MOS's extended batch features, it is possible to incorporate a context sensitive help feature into a batch process. This can be done by including a selected help hotkey, such as F1, within each KEY statement. Anytime the hotkey is pressed, a branch will be made to a portion of the batch file which will cause an appropriate help screen to be displayed. Note that if multiple KEY statements are used, multiple labels will be required at the start of the help screen processing logic.

An environment variable will be established to identify the help context of each area within the batch file. This will enable the help screen logic to determine which screen to display. For speed, this should be the name of a separate batch file to which a nested call will be made. Each help screen batch file will simply contain TEXT blocks with PAUSE commands between screens. To invoke such a batch file, simply place the reference to the environment variable in the position of a command. Refer to listing 4-11 for an example.

Listing 4-11

Example of context sensitive help feature:

```
:mainloop
...
set helpfile=help1
key %%a in (f1,a-e,cr) do call menu1%%a
goto mainloop
...
...
set helpfile=help2
key %%a in (f1,1-3) do call menu2%%a
...
...
:menu1f1
:menu2f1
%helpfile%
return
```

Although most examples of the KEY command's syntax (e.g. the PC-MOS USER GUIDE and the on line help facility) show the use of a variable %%A, the letter used here could be any other valid character. In addition, do not confuse this with the use of the special %A system variable which can be used to determine the current task number. To illustrate the difference, consider the following example:

```
KEY %%A IN (A-F) DO ECHO KEY PRESSED = %%A,
CURRENT TASK = %A
```

Compatibility

For the most part, batch files developed under PC-DOS will run unmodified under MOS. There are, however, some differences. If a batch file expects that naming another batch file within it, will not nest, but rather pass the chain of command onto the new batch file without any return to the original batch process, then some modification will be required for proper operation under MOS. Adding an ABORT command to the end of the second batch file will circumvent MOS's automatic batch file nesting feature.

NOTE: Effective with release 4.00 of MOS a new environment variable, $COMPAT$, has been added. It allows the user to modify MOS's behavior to allow for better compatibility with PC-DOS's batch file handling. See the section on "Using $COMPAT$".

If you wish to design batch files that take advantage of MOS's extended features but must also allow for the use of such batch files under PC-DOS then you should study the differences and similarities between these two systems carefully. The program ISMOS.COM, from the companion disk, can be used within a batch file to detect whether DOS or MOS is being run. ISMOS.COM will generate an ERRORLEVEL of 0 for MOS and 1 for DOS.

Some of MOS's batch file commands can be emulated through the use of utility programs. For example, the functionality of the AUTOCD command is self evident. It accepts a filename as a parameter and changes the current drive and directory to that which is recorded in the file. Other situations do not yield so easily to this type of cross system transla-tion. Implementing an equivalent for the KEY command would require "patching" into the internal operation of PC-DOS's command processor - not a trivial task.

Since it could be hazardous to forget, and invoke a MOS specific batch file when operating under PC-DOS, you might want to consider setting up some safety stop gaps. Using ISMOS.COM at the start of each batch file, to verify that MOS is the host environment, is one solution.

Another approach is to compensate or trap individual commands. For instance, if you create a batch file named KEY.BAT, when the KEY command is encountered while running under DOS, a batch file to batch file chain will be done -- with no return to the original process. This will not replace the function of the key statement but it will terminate the original batch process and prevent strange things from happening. KEY.BAT could contain a statement such as: "ECHO TERMINATED IN KEY.BAT" to notify the user of what has happened. Note that KEY.BAT should be kept in a directory which is within the command search path when running under DOS but not while running under MOS.

Another good candidate for this approach is the ABORT command. Creating ABORT.BAT (which need contain no more than an empty line) for PC-DOS use will provide a proper reaction to the processing an ABORT command within a batch file. As with KEY.BAT, this command replacement technique presumes that the ABORT internal command will be processed under MOS and ABORT.BAT will be found while running DOS.

Through the use of a certain trick, it is possible to construct a batch file such as XDEL.BAT so that the same file will work equally well under both MOS and DOS. XDEL is a type of batch process which makes use of a loop structure and the NEXT command to process a variable number of command line parameters.

However, under DOS, the SHIFT command is required in place of MOS's NEXT command. Rather than use the ISMOS.COM utility to branch one direction for MOS and another for DOS, you can simply create a pair of dummy commands.

The version of XDEL.BAT shown in listing 4-12 has been modified to incorporate this dummy command technique. Both the NEXT command of MOS and the SHIFT command of DOS appear within the batch process. What makes this work is that a file named SHIFT.COM exists within a directory which is only within the command search path when MOS is the host operating system. In the case where the system has been booted under DOS, a file named NEXT.COM exists. Again, this file is placed in a directory which only appears within the PATH statement when DOS is the host OS.

Each of these COM files contains just one instruction -- just enough to terminate the file. This results in a program file which is 1 byte in size and produces a null action. Therefore, when running under MOS, the NEXT command operates on the command line parameters and the SHIFT command does almost nothing. When running under DOS, the SHIFT command performs the required operation on the parameters and the NEXT command is just a null action.

To create SHIFT.COM and NEXT.COM, follow the list of instructions below. Wherever you see the symbol <cr>, press the Enter key.

1. At the command prompt, enter DEBUG NEXT.COM <cr> and ignore the message which says 'Error reading program into memory'. This message was produced since NEXT.COM doesn't exist yet.

2. Enter the command rcx<cr> (e.g. enter the letters r, c and x and then press the ENTER key)

3. Enter the number 1<cr>

4. Enter the letter a<cr>

5. Enter ret<cr>

6. Enter <cr> (just press ENTER)

7. Enter w<cr>

8. Enter q<cr>

9. Now, you're back at the command prompt. Since NEXT.COM and SHIFT.COM are both dummy commands, simply enter COPY NEXT.COM SHIFT.COM.

10. Place NEXT.COM in a directory which is in the command search path only when you are using DOS.

11. Place SHIFT.COM in a MOS only directory.

Listing 4-12

XDEL.BAT revisited - using NEXT and SHIFT.COM:

NOTE: comments enclosed in { }'s are not actually part of the batch file.

```
:loop
if (%1) == ( ) abort        {when %1 is null, all parameters processed}
echo deleting %1
del %1                      {do the deletion}
if errorlevel 1 goto delerror
next                        {2nd parameter --> %1, 3rd --> %2, etc}
shift                       {alternate to next}
goto loop                   {go back to :loop and check for more}
:delerror
echo error in delete command
```

Using $COMPAT$

There are two areas of batch file operations in which MOS's differences from PC-DOS can cause problems:

1. MOS nests batch file to batch file calls, while PC-DOS chains by calling the named batch file with no return to the calling batch file.

2. MOS sets an errorlevel on certain internal commands.

In order to allow the user to modify MOS's behavior in these two areas, a new environment variable, $COMPAT$, has been added effective with release 4.00.

For example, issuing the command:

```
SET $COMPAT$ = /N
```

will cause MOS to act like PC-DOS when one batch file is named from within another. If someone has a batch file which counts on PC-DOS's non-nesting type of behavior, they can use this feature. If, however, they have batch files which are designed to nest -- they must not use this option. This is primarily intended for compatibility with installation batch files which come with some applications.

Issuing the command:

```
SET $COMPAT$ = /I
```

will cause MOS to not set ERRORLEVEL on any internal commands. This is also intended for compatibility with installation batch files.

Any part of this command may be entered in either upper or lowercase, and the switches may also be combined. For example:

```
set $compat$ = /n/i
```

This feature control was implemented so that it would remain as set across command processor levels.

For example, if a particular $COMPAT$ setting is in place and the installation batch file does a COMMAND /C, or some other type of shelling, the switch options will survive.

The MAKESET Technique

In order to enter a string of data into the environment from within a batch file, a pre-written SET statement must be executed. However, there is nothing to stop one batch process from creating a new batch file, which happens to contain a SET statement, and then making a nested call to this new file. When invoked by the primary batch process, the MAKESET utility program obtains data from a specified source, formulates it into a SET statement and puts this statement into a batch file. Upon return from the MAKESET program, the primary batch process can then call the new batch file in order to execute the custom written SET statement it contains.

With this technique it is possible for a batch process to obtain data from a variety of sources, including a console prompt, a file, or an intertask pipe driver, and place this data in an environment variable. This increased capability brings MOS's batch processing to a level of sophistication which begins to approach that of a high level language. Some typical uses of this feature include user interface shells which can prompt the user for file names and other operating data, and list processing batch utilities which can read the contents of a text file and carry out a series of operations based on that data.

In order to clarify this idea consider the pair of batch files HELLO.BAT and HLTEMP00.BAT as shown in listing 4-13. HELLO.BAT is the primary batch file and HLTEMP00.BAT is the temporary file which was custom written by the MAKESET utility program when its console prompt was answered by a user named Pete.

For now, don't be too concerned by the exact syntax of the parameter line following MAKESET within HELLO.BAT. What is being done is to designate the type of data collection operation to perform, the name of the environment variable in which the data is to be SET and the name of the temporary batch file that MAKESET is to create.

Listing 4-13

Example of the MAKESET technique:

HELLO.BAT

```
echo Enter your name
makeset r envvar hltemp%a con 15
hltemp%a
echo Hello %envvar%
set envvar=
del hltemp%a
```

HLTEMP00.BAT

```
set envvar=Pete
```

In this example, the sequence of events would be as follows:

1. HELLO.BAT invokes the MAKESET utility, supplying the necessary parameters.

2. The MAKESET utility prompts the console for a string. A limit of 15 characters is placed on this string's length -- this being specified by the last MAKESET parameter.

3. The user answers the prompt with the name "Pete".

4. MAKESET forms the set statement SET ENVVAR=Pete and writes this to a file named HLTEMP00.BAT. Again, MAKESET was provided this name from a command line parameter.

5. After MAKESET terminates, a nested call to HLTEMP00.BAT is made from HELLO.BAT which causes the SET statement to be executed.

6. Control returns to HELLO.BAT where the environment variable is put to use.

7. The temporary file and the environment variable are then deleted for good housekeeping.

This example has been intentionally simplified so that the MAKESET technique can be introduced in a clear format. In actual practice there should additional statements to test for and respond to errors which could occur. Following the MAKESET command should be an IF ERRORLEVEL test to respond to any errors which this utility may encounter such as improper syntax in its parameter line or disk errors which occur during file I/O.

Also, the temporary file that MAKESET writes should include an IF ERRORLEVEL statement to handle the case where the SET statement fails due to insufficient free space in the environment. Remember that a nested batch file call does place a temporary ceiling on the environment's size so it is important to have already reserved enough room for the largest possible need with the ENVSIZE statement.

A MAKESET program can be written fairly easily in most any high level language, such as C, Pascal or BASIC, with a range of features limited only by your imagination. A sample, coded in C, has been included in the companion utilities disk.

CHAPTER 4

The RJE Technique

The term RJE is an acronym for Routed Job Entry, a type of batch processing used primarily in mainframe and minicomputer data processing installations. Running a batch file under MOS is somewhat similar. When you enter a batch file's name at the command prompt you are submitting the job for processing by MOS's $$SHELL.SYS.

One consequence of submitting a batch file in this immediate mode is that you are tying up your task. You could, of course, just PAMswitch to a spare background task and run the batch file from there. While there is certainly no reason this can't be done, there are also cases where it is more convenient to have this process automated. Supporting such a feature involves setting up a background task dedicated to watching for batch files which have been submitted to it for execution.

A "watcher" task can be especially useful when a MOS system is remotely accessed through a modem. When a transfer of files is involved, a watcher task may be employed at the host machine to archive files before downloading and to un-archive ones that have been uploaded. Through the use of an RJE subsystem, the remote user can invoke these archiving processes at the host by simply submitting small batch files to an RJE watcher task running on the host. This will result in shorter transfer times and lower phone connect charges.

An RJE task can also be useful to gate access to a non-multiuser application. When a number of users access an application through the RJE method there is actually only one user at a time, as far as the application program is concerned. The application must, of course, be amenable to being operated completely from a batch file rather than requiring interactive input. The RJE feature can also make it easy to utilize a network file server for functions such as sorting, report generation, compilation of programs, etc.

Watching for an RJE File

A general purpose approach is to dedicate a task to perpetually watch for the submission of batch files and process them when found. Adding a task whose startup batch file contains a loop as shown in listing 4-14 would do the job. However, this implementation is not efficient. The continuous looping of this batch file would produce a small load on the system's throughput. The ideal situation is for the task in which this watching is being done to be suspended for the majority of the time until a batch file is detected.

Listing 4-14

```
:recycle
if not exist rjejob.bat goto recycle
rjejob
del rjejob.bat
goto recycle
```

The method illustrated in listing 4-15 comes a little closer to this ideal. This method dilutes the process of checking for an RJE file. The command 'MOS WAIT 10' is a use of the system utility MOS.COM which forces the task into suspension for 10 seconds after each IF EXIST test. Running this RJE operation from a RAMdisk will also further reduce the processing overhead that the RJE task will impose on a system.

Listing 4-15

```
:recycle
mos wait 10
if not exist rjejob.bat goto recycle
del rjejob.bat
goto recycle
```

Although incorporating a wait period will result in a slower response time to submitted jobs, this shouldn't be too much of a problem within this context. Batch submitted jobs don't tend to be of the high priority type -- those types of jobs are run right away in your current task. In selecting a timing value proper balance must be considered. Too short of a value loads the system and, one that's too long will delay the detection of each job.

It is still possible to do better. Extra processing overhead is still being incurred every ten seconds to process the lines of the batch file. Not only is a disk access required to test for an RJE job file, but disk accesses must be made repeatedly to read in the contents of the watcher batch file.

This further optimization can be made by using a small utility program to check for RJE files. Such a program can use a system call to tell MOS, "I surrender my processing time until X timer ticks pass or a key is pressed". This keeps watcher's system load as light as possible since the only overhead incurred is the test for the file. An example watcher utility is included on the companion disk.

Writing an RJE Batch File

When designing a batch file designed to run under an RJE system some additional considerations must be made. An RJE run batch file will often have a different "perspective" than a batch file designed to run in your current task. This is especially true when the RJE task exists on a different computer system than the one from which the file will be submitted. You must consider what utility programs are available to the RJE task and whether they can be found through the command search path for the RJE task or whether an explicit path will be needed. In each system where the RJE technique is to be used, a documentation file should be written which describes pertinent details of the RJE task such as the default drive and directory, the command search path and what programs are available.

If your RJE batch process will change the current drive and/or directory, it should save the current ones for later restoration with the AUTOCD command. In addition, do not use ABORT -- an RJE batch file runs nested from the file RJE.BAT (which is probably nested from the RJE task's startup batch file). If an exit method is needed other than coming to the end of the file, use the STOP command. Also, be sure that any temporary files are made task specific by including the %A variable in the filespec.

When designing an RJE batch file, it is important to consider the order of events you are invoking, especially if irreversible file operations are involved. If you are going to submit an RJE batch file to unarchive a file you've just uploaded from a remote workstation, don't run the uncompression utility and then just erase the archive file. If the server's disk becomes too full midway into the expansion, you wouldn't want that error to result in the loss of the archive file. In this case, you would want to clear some disk space and restart the job. If the delete was allowed to happen from the batch file, you would have to upload the archived file again. A sample RJE batch file is included on the companion disk.

The Flag File Technique

As explained in Chapter 3, MOS cannot automatically manage access to all system resources. This is especially true of features which are present in your system through the installation of add-in expansion cards. Some examples are adapter cards for data acquisition and special purpose communications systems such as FAX and mainframe terminal emulators. Since these types of add in hardware features are usually not designed for simultaneous access by more than one user, some means of access control must be implemented to prevent confusion. This situation can also exist if you wish to allow multiple users to share a single-user type of program (e.g. an accounting package or simple data base).

One means of implementing an automatic form of arbi- tration is to use a flag file technique. For each item to be controlled create a file with a corres- ponding filename and keep all such files in one common, dedicated directory. For example, suppose the following files have been placed in C:\FLAGDIR:

```
FAX.___
SCANNER.___
ACCTNG.___
```

You may create these files with a simple statement such as "ECHO AFAX.___". It does not matter what data is contained within the file as it is the file's filename and extension that are significant. Each file extension is initially composed of three underscore characters to indicate that the corresponding item is not currently reserved. When a user needs to reserve an item, they will use a batch file to change the extension. This batch file, LOCK.BAT, is shown in listing 4-16. To use this file, specify the item's filename as the first parameter. For example: LOCK SCANNER.

Listing 4-16, LOCK.BAT

```
if (%1) == ( ) goto noparm
if %1 == ? goto show
cd  > c:\flagdir\flagfl%a
c:
cd \flagdir
set lockerr=
ren %1.___ %1._%a
if errorlevel 1 goto resvfail
\mosbin\filemode +a %1._%a >nul
echo Reserving %1 for task %a
:end
autocd c:\flagdir\flagfl%a
del c:\flagdir\flagfl%a
abort

:resvfail
echo Failure in reservation attempt for %1
set lockerr=1
goto end

:show
dir c:\flagdir
abort

:noparm
echo.
echoUsage: LOCK itemname        Locks an item
echo.
echoLOCK ?                      Shows reservations
```

The primary function of this batch file is to attempt to RENAME one of the flag files so that its extension is changed to reflect the current task number. This is done through the use of the %A system variable.

The RENAME command will only succeed if the particular file is not already reserved; in other words, if its extension is "___". In the event that two users attempt to reserve the same item at the same time, only one will succeed; the one that was actually first in the time slice order of execution. The other user will receive an error message from LOCK.BAT.

Of course, with the ability to reserve an item must come the ability to free it for use by others. The batch file for this, UNLOCK.BAT, is shown in listing 4-17. Note that by using a "?" as the parameter for either LOCK.BAT or UNLOCK.BAT, you may review the current status of all items which are tracked by this technique.

Listing 4-17, UNLOCK.BAT

```
if (%1) == ( ) goto noparm
if %1 == ? goto show
cd  > c:\flagdir\flagfl%a
c:
cd \flagdir
ren %1.* %1.___
if errorlevel 1 goto relfail
\mosbin\filemode +a %1.___ >nul
echo Releasing %1
:end
autocd c:\flagdir\flagfl%a
del c:\flagdir\flagfl%a
abort

:relfail
echo Failure in release attempt for %1
goto end

:show
dir c:\flagdir
abort

:noparm
echo.
echoUsage: UNLOCK itemname          Unlocks an item
echo.
echo          UNLOCK ?              Shows reservations
```

Upon close inspection of these batch files you may notice two things whose meaning may not be self evident. The first is the use of MOS's FILEMODE command.

After the flag file is renamed, FILEMODE is used to force the file's archive bit to be set. This is done to insure that this file is included in the next disk backup operation. (Since renaming a file does not change the file's contents, RENAME does not set the archive bit.) Not only will your system's reservation status survive brown-outs and system shutdowns, but disk restorations as well.

The second thing which bears explanation is the use of the environment variable LOCKERR. LOCKERR will be SET into the environment in the event of a reservation failure. This is done to provide a signal to another batch file which might be making a nested call to LOCK.BAT. An example of this follows.

This reservation method will only work if all users cooperate and remember to lock an item before using it, and unlock it when through. Where applicable, you could enforce adherence by gating access to applications through other batch files. For example, if the fax system is operated by the program FAX.COM then you could create RUNFAX.BAT as shown in listing 4-18. By requiring all users to access this device through the RUNFAX batch file, access control will be fully automated.

Listing 4-18

RUNFAX.BAT - an example of automating the process:

```
lock fax
if not (lockerr) == ( ) goto faillock
...
...
fax
...
...
unlock fax
abort

:faillock
set lockerr=
echo Sorry, FAX is presently locked
```