intel®

APPLICATION NOTE

AP-245

October 1986

Using Command Files to Speed Program Development

SRIVATS SAMPATH
DSO APPLICATIONS ENGINEERING

Order Number: 231481-001

INTRODUCTION

Recently, the computer industry has leaned toward providing a very friendly interface between human and machine:an interface that allows the user to be more productive in the least time possible, an interface that gives him or her the ability to use all of the computer's advanced features. Tools to assist the user include the command line interpreters (CLIs), HELP texts, and command file capability.

Recognizing the need for a sophisticated human interface, Intel has provided the advanced command line interpreter and syntax driver, HELP texts, and submit file capabilities on the Series IV Microcomputer Development System. This application note deals with the power and use of the Series IV command file capabilities. These facilities, available only on the Series IV and the network resource manager (NRM), represent a major improvement over the Series II and Series III systems.

Command files are files that can be executed by the host system. They are not programs. Typically, the computer recognizes the keystrokes from the keyboard and executes the operation selected. However, this becomes time-consuming for repetitive tasks involving many keystrokes. Constant user interaction is needed during the whole execution cycle, i.e., as soon as one operation is complete, the user has to type in the next command. If all these commands could be put in a file, and if the computer could read this file and execute all commands sequentially without any user interaction, we would have a system that drastically increases user productivity and reduces user fatigue.

Command files may be executed using two commands:SUBMIT and EXPORT. SUBMIT executes the command file instantaneously, while EXPORT sends the command file for execution at another system at a time determined by the DJC manager. (Refer to Application Note, AP-244, "DJC - The Key to Increased Network Productivity").

The SUBMIT command on the Series IV allows the user to replace commands typed in from the keyboard with ommands from a file. The submit command redirects console input to the specified file. The Series IV also has a utility editor called BATCH that helps the user in creating a submit file. BATCH is an editor that incorporates within it the Series IV syntax driver. The syntax driver is a human interface that keeps prompting the user for correct variables and options. With the BATCH utility, a user can create a submit file with no difficulty.

Command files can also be described as pseudoprogramming languages. These files can execute and perform logical operations, file I/O, support memory variables, looping, repeat function, and parameter passing. These commands execute their functions in a simple but effective way. They should be used with all other commands for more effectiveness. Command files resemble a very functional interpretive language. This is a very powerful utility on the Series IV and can be used to perform a variety of applications without requiring long programs to be written.

To illustrate these multiple features, this Application Note will describe a command file called MAILMAN, which is an internetwork mail utility. MAILMAN has the capability to mail across multiple networks using only existing software. The Series IV command file capability has been used extensively in this application.

COMMAND FILES AND THEIR CONSTRUCTS

This section describes all the command file operators, their functions and the interrelationships between them.

The LOG Command

One of the commands often used with a submit file is the LOG command. LOG copies all console output to the file specified.

For example, type this at the command line:

```
>LOG TEST.LOG           ;also send console output to TEST.LOG
>dir /APS1/USER.DIR expanded
iNDX-W31 (V2.8) DIR V2.8
DIRECTORY OF /APS1/JSER.DIR

FILE--NAME     OWNER--NAMEFILE--LENGTH  TYPE OWNER--ACCESS
JOHN.DIR       JOHN           2048      DIR  del dis add
CHRIS.DIR      CHRIS          4096      DIR  del dis add
SRIVAT.DIR     SRIVAT         2048      DIR  del dis add
NORI.DIR       NORI           2048      DIR  del dis add
WAYNE.DIR      WAYNE          4096      DIR  del dis add
WORLD.DIR      WORLD          2048      DIR  del dis add
SUPERUSER.DIR  BRIAN          2048      DIR  del dis add

total bytes used:   44021
>log :bb:                ;stop additional redirection to file
```

The LOG file will contain the exact output of the DIR command. We have selectively written specific data into a file that we will use later. This feature is very useful in command files and will be seen in MAILMAN.

In any system, the ability to pass parameters or variables from one program to another is very important. By passing parameters, a command file can be made to do a variety of tasks. The Series IV command file structure allows the passing of up to 10 parameters from the command line. These are designated %0 through %9.

For Example:

```
#CC86 %0.c debug %1

IF %status = 0 THEN

     link86 %0.obj &
     l/%1.lib,      &
     l/%2.lib,      &
     l/bvcslb.lib,  &
     l/87null.lib,  &
     to %0.86       &
bind            &
ss(stack(+800h),memory(+2800h))
END
```

The user locally invokes this command file by typing in:
```
>SUBMIT
COMPILE(CHECKEXIST,SMALL,SCLIB)
```

When the command file is executed, it substitutes:

CHECKEXIST	for %0
SMALL	for %1
SCLIB	for %2

The user can therefore, have one command file that can be used to compile and link different sources with different libraries just by specifying them at the command line level. This parameter-passing feature provides increased command file versatility. Without this feature, the user would have numerous command files, each executing a specific operation. This feature gives the user substantial flexibility and helps reduce the time to develop unique files for each application.

Command Line Variables

Command files also support the assignment of variables to alphanumericstrin gs through the SET command. SET assigns the value on the right to the variable name on the left.

For example:
```
SET NAME TO %0
```

will set the contents of %0 passed in from the command line to the CLI variable, NAME. Any further reference to %NAME will yield the value of %0. %NAME will access the contents of the variable, NAME. The Series IV CLI has an undocumented built in command called "DUMP". DUMP aids in debugging command files. It displays all the variables in a command file and their corresponding values.

For Example:

On submitting a file TEST.CSD

```
>SUBMIT TEST                   will yield
>SET NAME1 to ""WAYNE''        ;set name1 to wayne
>SET NAME2 to ""SRIVAT''       ;set name2 to srivat
>SET NAME3 to ""JOHN''         ;set name3 to john
>DUMP                          ;show all variables and their values
NAME:""STATUS'' VALUE:""0''
NAME:""NAME1'' VALUE:""WAYNE''
NAME:""NAME2'' VALUE:""SRIVAT''
NAME:""NAME3'' VALUE:""JOHN''
EXIT COMMAND FILE /APS1/USER.DIR/SRIVAT.DIR/APNOTE.DIR/TES.CS
```

The system supports a predeclared variable called STATUS, which is set by the DQ$EXIT value of a previously executed program. For example, a successful termination will normally set STATUS to 0, while an error condition will return another value. For example, consider the UDI call DQ$EXIT(VALUE).

A dq$exit(0) will set STATUS to 0

A dq$exit(1) will set STATUS to 1

The value of STATUS depends on the parameter passed by the existing program. This tool is very useful for conditional compiles and conditional links. The command file can link and locate by monitoring STATUS only if the program is compiled without any errors.

Example:

```
cc86 %0.c debug
if %status = 0 then              ;link only if compile successful
     link86 %0.obj,        &
     l/sqmain.obj,         &
     l/sclib.lib,          &
     l/small.lib,          &
     l/bvcslb.lib,         &
     l/87null.lib          &
     to %0.86              &
     bind                  &
     ss(stack(+800h),memory(+2800h))
end
```

The command file above will link the object files only if the compiles are successful. This will save the time of whole like cycle without having the correct objects.

Most Intel-supplied software, especially the translators and utilities, use this concept. A successful program completion will exit with STATUS set to 0, and a program abort or termination will exit with STATUS set to something other than 0.

Accessing Data Files

The file I/O capabilities of command files are very powerful. The commands for file I/O are OPEN and READ. For example, consider the LOG file FILE.TMP, generated by the sequence:

```
LOG FILE.TMP
DIR /
LOG :BB:
```

FLE.TMP will contain:

```
>dir /
iNDX-W41 (V2.8) DIR V2.8
DIRECTORY OF /
FILE--NAMELOCATION  ACCESSIBILITY
APS--WO       remote
W1            remote
APSO          remote
APS1          remote
SYS             local
>log :bb:
```

A command file is shown that accesses this LOG file to discover the volume root name for the network:

```
OPEN file.tmp
COUNT 14
     read skip ;skip over the first 14 words of the file
     end
read root
end
```

In a DIR / command, the first file name is the volume root name. In this case, APS—W0 is the system volume root name and has to be assigned to some variable for future use. The command file utility OPEN for file I/O is used to gain access to the file FILE.TMP. The COUNT command is used as a loop counter that will loop around the number of times specified.

In this case, COUNT 14 will loop 14 times. Each time, it will set the variable SKIP from one word in the LOG file. A Word can be defined as a set of characters separated by a white space. We effectively skip over one word at a time. In this case, APS—W0 is the 14th word from the start of the file. So, the loop will skip 13 words and then read the system volume root name into the CLI variable ROOT. The COUNT, READ, OPEN and SKIP commands, built into the CLI, can be used only in submit files. The READ ROOT command will read the 14th word into the memory variable ROOT. %ROOT will contain the value of ROOT, which is, in this case, APS—W0.

Only one file can be opened at a time. There is no explicit CLOSE function. Opening another file will close the previous one. To force the Close of a parameter file, use the OPEN command on a nonexisting file. A combination of the LOG, OPEN, READ and SKIP commands help in doing very functional but effective file I/O

Conditional Command File Execution

Since the Series IV command file is like a pseudo-interpreter, it also supports logical operations such as IF, THEN, ELSE. The example below highlights how these constructs can be used within a command file.

This command file compiles any C program and then checks for a successful compile. If the compile is successful, the command file proceeds with linking and binding. If the compile is not successful, the file is an error reporter. More information on REPORT can be obtained from AP-244, an application note on distributed job control.

```
cc86 %0.c debug                         ;Compile the program
if %status 0 than                       ;If error in compile
    REPORT Error in compile of %0.c     ;Send message to user
else                                    ; and exit.
            REPORT Successful Compile. Proceeding with LINK
        link86 %0.obj,      &
        1/sqmain.obj,       &
        1/sclib.lib,        &
        1/small.lib,        &
        1/87null.lib        &
        to %0.86            &
        bind                &
        ss(stack(+800h),memory(+2800h))
if %status = 0 then                     ;Check for error in link
    REPORT Successful Link. End of Job. ;If no error inform user
else
    REPORT Error while linking.....     ;If error inform userand
end                                     ;and exit.
```

Command File Looping

The COUNT construct is a looping control. REPEAT is an additional construct and works in conjunction with the WHILE and UNTIL commands. REPEAT will loop until the condition specified by the WHILE or UNTIL command is satisfied.

```
REPEAT
     UNTIL %STATUS = 2
          any operation
END
OR
REPEAT
     WHILE %STATUS <> 2
          any operation
END
```

These logical operators can be used in any combination as long as the syntax is correct. Each loop should have a matching end statement.

MAILMAN (A BRIEF EXPLANATION)

MAILMAN is a command file that allows users on one network to send mail to users on another network over an Ethernet cable. The network users do not have to distinguish or remember the USER/NRM configuration. MAIL is used as normal, and MAILMAN running as a background task knows the configuration and behaves accordingly. The MAILMAN utility uses existing software and the powerful constructs of the Series IV command file utility to illustrate that complex problems can be solved simply.

In a typical multiple network environment, communication between users on one network with users on the other network is very important. Since electronic mail supports only one network, there was a need for a system that supported mail over multiple networks. Writing a program in one of the high-level languages or assemblers using Ethernet protocols would require substantial time for designing, developing, and debugging.

The MAILMAN utility is an example of how command files increase productivity and help solve complex applications. MAILMAN, which uses almost all the commands and constructs supported by command files, will help the reader understand how command files can be used for any particular application.

In this example two NRMs will send mail between each other by executing the MAILMAN utility on import stations at both ends. These import stations import from a utility queue called iNDXUTILITY.Q. For more information on queues and remote job execution, refer to Application Note AP-244 titled DJC:Key to Increased Network Productivity.

MAILMAN generates several data files using the LOG command during execution to discover the various system variables. It also depends on two data files called REMOTE.USERS and LOCAL.USERS. These files have the sameformat and are used to distinguish which NETWORK each user is Sysgenned onto. The format is:

```
Line 1:NRM root volume name
Line 2:Username
Line 3:Username
•
•
•
Last line:blank <to signify the end of
list>
```

These data files should be placed in the directory MAIL.DIR of each system.

For example, the file LOCAL.USERS under /APS—w0 will look like:

```
APS--WO
SRIVAT
WAYNE
JOHN
CHRIS
<blank>
```

And the file REMOTE.USERS will look like:

```
PMO
PAUL
FRANCIS
STU
SUNIL
<blank>
```

Each of the REMOTE users have to be sysgenned onto the local network as users but without a home directory. The user MAILMAN has to be sysgenned intothe network as a user with a home directory.

MAILMAN (THE COMMAND FILE)

```
;****************************************************************************
;* CHECKMAIL.CSD . This is a submit file that allows multiple NRM mail.     *
;* A detailed explanation of the system requirements is in the CHECKMAIL.DOC*
;* file. CHECKMAIL allows users on one NRM to mail messages to users on     *
;* other NRMS. There is no limit to the number of NRM's, but you must read  *
;* the Toolbox manual or CHECKMAIL.DOC to effect modifications for more than*
;* two NRMs.  This file is set up for two NRMs only.                        *
;****************************************************************************
;****************************************************************************
; Need to know the root volume name
;****************************************************************************
log file.tmp
dir /
log :bb:
open file.tmp
count 14
    read skip
    end
read root
open file2.tmp                           ; To close param.file
;****************************************************************************
; If the user did not supply a parameter use their USER NAME
;****************************************************************************
if %Ø <> ""
    set name to %Ø
else
    log file.tmp ; Who is currently using this command file
    id
    log :bb:
    open file.tmp
    read skip,skip,skip,skip,name
    end
;****************************************************************************
; If the user is MAILMAN then we are in receive mode
; else we are in transmit mode
;****************************************************************************
if %name <> MAILMAN then
; Can now check for mail
    delete message.found
    mail box %name
    save 1 message.found
    q
; Check to see if there was a message in the mailbox
    checkexist message.found
    if %status = 1
        set sent to false
        open /%root/mail.dir/remote.users
; Read the root volume of the remote network
        read rroot
        repeat
            while %sent = false
```

231481-1

```
; Read one of the remote system user names
            read remote
            while %remote <> ""
            if %name = %remote then
                open file2.tmp
                log file.tmp
                time
                log :bb:
                open file.tmp
                count 11
                    read skip
                    end
                read time
                nncopy message.found to %rroot/mail.dir/%name/%time    &
                    username (mailman ) password (post) nrm (Ø)
;***************************************************************************
; The message has been forwarded to the other system,
;     remove it from the local mailbox
;***************************************************************************
                mail box %name
                delete 1
                e
                set sent to true
                end
            end
        if %sent = false then
; This is a local user message
            Report You have mail in box %name
            end
        end
;***************************************************************************
; The user MAILMAN, operate in receive mode
;***************************************************************************
else
    set name to user
    repeat
    open /%root/mail.dir/local.users
; Ignore the root name
    read skip
; Skip to the user name
    read %name
    while %user <> ""
    log file.tmp
;***************************************************************************
; Check for 'timed' mail delivery from another system
;***************************************************************************
    dir /%root/mail.dir/%user for ??:??:??
    log :BB:
    open file.tmp
    count 16
        read skip
        end
    repeat
        read file
        while %file <> ""
```

231481-2

```
107 ;************************************************************************
108 ; A 'timed' message has been found, MAIL it to the appropriate user
109 ;************************************************************************
110         mail /%root/mail.dir/%user/%file to %user &
111         subject(Arrived at %FILE and forwarded by Mailman)
112         delete /%root/mail.dir/%user/%file
113         end
114     set name to "skip,%name"
115     end
116 end
117 ;************************************************************************
118 ; All done, reinvoke myself
119 ;************************************************************************
120 export /%root/checkmail(%0) to iNDXutility.q nolog
```

231481-3

MAILMAN (AN IN-DEPTH LOOK)

Many comments, included to help your understanding of the program flow, could be edited out to speed execution. There are some concepts used in this program that need additional explanation.

#65

We use the NRM to NRM communications package discussed in Application Note AP-241 (Multiple NRM Ethernet Communications) to communicate between the NRMS. If other NRMs were not on the same Ethernet cable, we could change this line only to incorporate autodialing to other NRMs.

#114

"SET NAME TO "SKIP", %NAME"

As described earlier, the CLI can have only one file open at any time. The opening of any other file will close the previously opened file. In line #88, we open the file LOCAL.USERS and skip to the first name. Once this user name has been established, we need to check for mail in his or her Mail directory. This operation is done in lines #99 to #114. However, we now need the name of the next user. Since we already closed the file LOCAL.USERS, the next time lines #83 through #94 are executed, the file pointer will point to the same name and will repeat the loop. By setting NAME to "SKIP, %name", the next time the file is opened, it will automatically read the second name. The CLI variable %NAME will be "SKIP, %NAME" and the command READ will skip one word and read the next. The third time, NAME will be "SKIP,SKIP, %NAME", the fourth time, "SKIP, SKIP, SKIP, %NAME". Even though the file LOCAL.USERS is constantly opened and closed, our file pointer is still intact and points to the correct word.

#120

"Export /%root/CHECKMAIL(%0) to indxutility.q nolog"

We need this file to be executing forever:checking the mail and sending it to the right networks. This is an example of looping in export files, i.e., having a remote job run forever. For more information, refer to the Application Note AP-244 DJC, Key to Increased Network Productivity.

#44

CHECKEXIST.86 is a file checker. If the specified file exists, it will exit with STATUS set to 1. Otherwise, STATUS is set to 0. This is used to determine the existence of a file. This small utility was developed for use within the MAILMAN package.

Example:

```
CHECKEXIST TEST.FILE will
set STATUS to 1 if TEST.FILE exists
else
set STATUS to 0
```

Within the MAILMAN utility, CHECKEXIST is used to check if any mail messages exist for the user specified. Looking at the source of CHECKEXIST.C shown in Appendix A, the concept of STATUS becomes very clear.

CONCLUSION

MAILMAN is an extensive example of the power of the Series IV command file capabilities. The complete file was developed and debugged in less than a week - far shorter than writing an application program to talk over Ethernet. The Series IV command file capability enables you to build upon software you already have to reach higher heights more quickly.

APPENDIX A (CHECKEXIST.86)

```
/************************************************************************/
/*  CHECKEXIST.C            Existence checker for files.                */
/*  CHECKEXIST will exit with an exit code of "Ø" if file is not found */
/*  else it will return with an exit code of 1. This will be passed    */
/*  into %STATUS in a command file.                                     */
/*  Syntax for CHECKEXIST.86 :                                          */
/*  CHECKEXIST < filename >                                             */
/*                                                                      */
/*  Example:                                                            */
/*                                                                      */
/*  The batch file CHECK.CSD contains these statements.                 */
/*  CHECKEXIST My.File                                                  */
/*                                                                      */
/*  Submitting this batch file will set %STATUS to Ø if file does not  */
/*  exist and 1 if file exists.                                         */
/*                                                                      */
/*  For an example of the use of this CUSP see the Mailman example for */
/*  multiple network mail.                                              */
/************************************************************************/

#include <:f2:stdio.h>
#include <:f2:ctype.h>

/************************************************************************/
/*   Start of main routine                                              */
/************************************************************************/
main(argc, argv)

int       argc;
char     *argv[];

{

    FILE *fp, *fopen();

    if (argc == 1)
    {
        puts("Error..Filename not specified.");
    }
    /* Exit with return value Ø if file does not exist */
    /* else exit with a 1                              */
    if ((fp = fopen(*++argv, "r")) == NULL)
    {
        dq$exit(Ø);
    }
    else
    {
        fclose(fp);
        printf("FOUND FILE : %s\n",*argv);
        dq$exit(1);
    }

}
```

231481-4