RL-80-008

Rutherford Laboratory
CHILTON, DIDCOT, OXON. OX11 0QX

RL-80-008

An Introduction to CMS on the RL Coupled System

D Asbury, R Freeman, T Pett and S Ward

March 1980

RL-80-008
First edition

AN INTRODUCTION TO CMS ON THE RL COUPLED SYSTEM

David Asbury, Robin Freeman, Tim Pett, Sue Ward

Rutherford Laboratory
Chilton
Didcot
Oxon, OX11 0QX

March 1980

An Introduction to CMS on the RL Coupled System

CONTENTS

P R E F A C E

This manual is intended as a first guide to VM/370, CP and CMS as implemented on the IBM 3032 at Rutherford Laboratory. It describes only the simplest features of these systems and Rutherford modifications and additions to them. There are many facilities which are described only briefly in this manual. In these cases there are references to other manuals where full details may be found, but the reader should note that some facilities described in IBM documentation are not implemented at Rutherford.

This manual covers the simple features of file creation and editing under CMS, running programs under CMS and CMS batch, and job submission to the MVT batch systems on the IBM 360/195's. For many users this may be all the documentation which is required. For those who wish to make use of other facilities, there are frequent references to other manuals, particularly the IBM CMS User's Guide and the RL VM User's Reference Manual. The CMS User's Guide gives a very readable account of the CMS system but does contain a lot of information which is not relevant to users of the system at Rutherford. The User's Reference Manual briefly describes all CP and CMS commands which are available at Rutherford with full details of any commands which have been modified, enhanced or added locally.

Chapter 1: INTRODUCTION

1.1 What is VM/370?

VM/370 is the system control program running on the IBM 3032 at the Rutherford Laboratory. It allows many virtual machines to share the resources of the physical machine concurrently. A virtual machine simulates a real computer and its I/O devices including a virtual CPU, virtual storage, a virtual printer, card reader and card punch, etc.. Each terminal user has a virtual machine and the user's terminal becomes the virtual operator's console for that machine. Each virtual machine runs its own operating system which for most users will be CMS.

There may be other virtual machines dedicated to special purposes. For example, one virtual machine runs as a CMS batch machine; another currently runs an MVT batch system as a back-end for the coupled system.

1.2 What is CP?

The Control Program (CP) is the part of VM/370 that controls the resources of the real machine. It also controls the creation of virtual machines, communications between them and communications between a virtual machine and the real machine.

VM/370 has a directory containing the user identifier (userid) and password for each registered user. When a user logs on, CP creates a virtual machine for that user with virtual storage and virtual devices as defined in the VM directory.

CP has a command language which allows the user to control the operation and configuration of the virtual machine and to communicate with other virtual machines. These commands are entered at the user's terminal which is the virtual console of the virtual machine.

1.3 What is CMS?

The Conversational Monitor System (CMS) is an operating system which can only run in a virtual machine. It gives the user facilities for creating and manipulating files and for compiling, testing and debugging programs interactively in the same virtual machine. CMS can be loaded with an IPL (initial program load) command to CP but at Rutherford this is done automatically when a user logs on.

A virtual machine which is to run CMS must have at least 320K bytes of virtual storage of which 128K is used by the CMS nucleus. At Rutherford the default machine size is 320K. This may be altered by a CP command up to a maximum of 512K.

Each registered user is issued with a portion of disk space known as a minidisk on which the user's files are stored. These minidisks reside on IBM 3350 disks and must be allocated in units of one cylinder. One cylinder contains approximately 570K bytes of storage space.

1.4 What is CMS batch?

The CMS batch facility allows the user to run programs in a special virtual machine which is dedicated to running batch programs only. These programs run sequentially with only one program in the machine at a time. However, there may well be more than one CMS batch machine, perhaps each dedicated to a different job class.

The CMS batch machine should be used for programs which require too much virtual storage or use too much CPU time to be run conveniently in the user's own virtual machine. In particular, the FORTRAN H compiler requires more than 512K of storage and must be run in CMS batch.

1.5 What is MVT batch?

OS/MVT (Multi-programming with a Variable number of Tasks) is the operating system which runs on the IBM 360/195's and currently within a dedicated virtual machine on the IBM 3032. It allows several jobs to run concurrently within the same CPU. Jobs may be submitted to the MVT batch machines from CMS.

There are two restrictions on programs running under CMS which make it essential for such programs to be run in MVT batch. The first is that OS data-sets can be read but not written from CMS. The second, which is imposed at Rutherford for operational reasons, is that CMS programs are not permitted to access tapes.

Jobs run in MVT batch are controlled by OS/360 Job Control Language (JCL) and all the relevant JCL must be contained in the CMS file which is to be submitted to MVT. To facilitate the parameterisation of JCL within a CMS file and the concatenation of several files, the PLANT/SUPPLY system (also available on the Data Editing 4080), has been implemented under CMS.

For a full description of how to use MVT JCL see the Computer Introductory Guide And Reference (CIGAR).

1.6 The Rutherford Configuration

The current configuration of the IBM 3032 and two IBM 360/195's is shown in Figure 1. The CEM MVT batch system runs in a virtual machine under VM/370 in the 3032 together with the machines running CMS and CMS batch. The front-end software, HASP, MAST, DKNCP and ELECTRIC, runs in the FEM MVT system on one of the 195's. The version of HASP in this machine is large because it contains the code to control all the remote work-stations. The other 195 runs another MVT batch system.

At present, only directly connected IBM 3277 terminals (or their equivalents) and terminals attached to the Gandalf PACX switching device have access to CMS. The intention is to move the FEM MVT system into a virtual machine on the 3032 allowing both 195's to operate solely as batch machines. When this has been achieved the control of the work-stations will be transferred gradually from HASP to VNET which is the VM equivalent. This will give work-station terminals access to CMS and should produce a considerable performance improvement on the 3032.

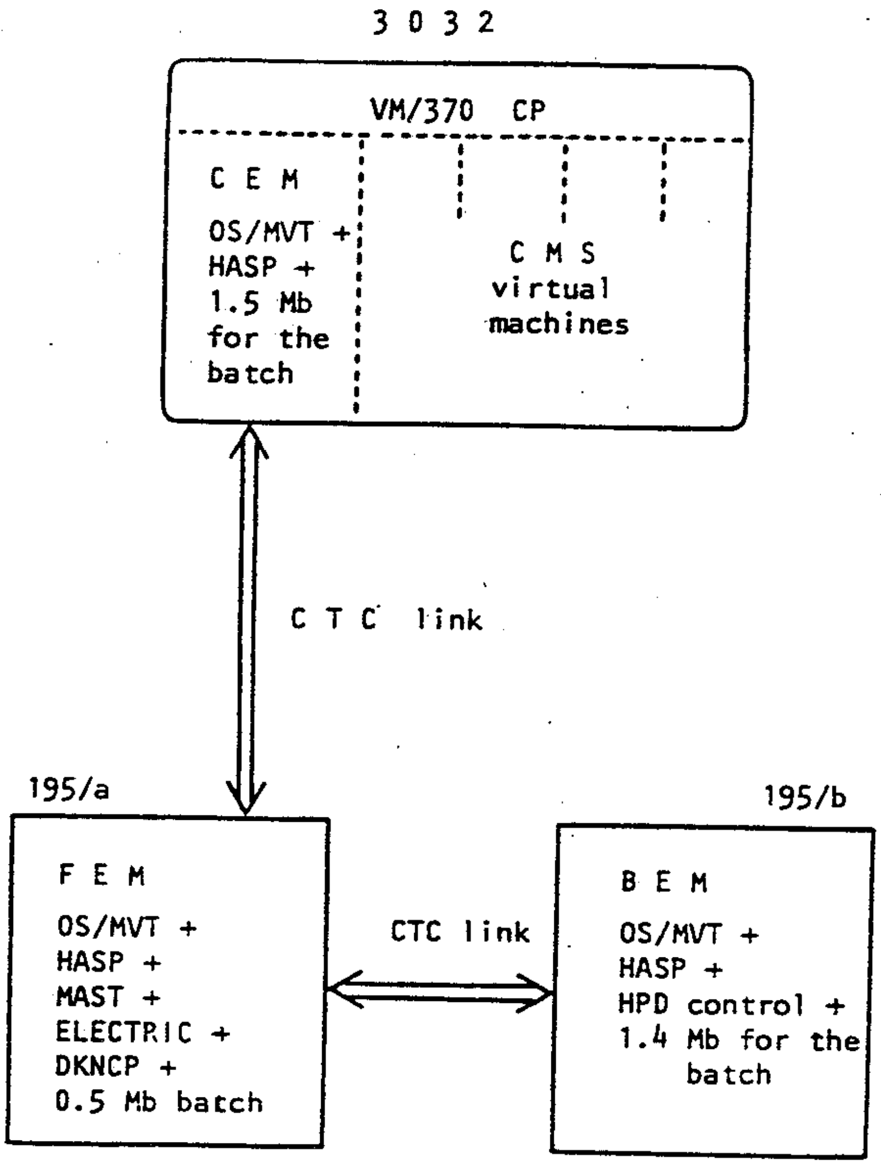
3 0 3 2
VM/370 CP
C E M
OS/MVT + HASP + 1.5 Mb for the batch
C M S virtual machines
C T C link
195/a
F E M
OS/MVT + HASP + MAST + ELECTRIC + DKNCP + 0.5 Mb batch
CTC link
195/b
B E M
OS/MVT + HASP + HPD control + 1.4 Mb for the batch

Figure 1: RL Coupled System

Chapter 2: HOW TO GET STARTED

2.1 Allocation of Identifiers

To register as a user, you will be asked to provide your name, your 195 identifier and account number, and the normal destination of your output. You will also be asked to select a login password.

You will be issued with a userid and a mini-disk (this will be your A-Disk - see the description of Filemode in Chapter 3). Your mini-disk will have 2 passwords associated with it, a READ password and a READ/WRITE password; these are for use by other users wishing to access your disk, and it will be up to you how many people know your passwords. Each of your three passwords may contain from 1 to 8 characters. If possible, your userid will be your initials.

2.2 Use of Terminals

For most users access to CMS will be from ASCII terminals connected to the 3032 either through the Gandalf PACX or through a workstation.

2.2.1 Access via the Gandalf PACX

There are currently 9 ports available; the number to dial is 02. Having done this you will receive the reply

```
SERVICE 02 START              <-    From PACX
VM/370 ONLINE                 <-    From VM/370
```

You should now hit the ENTER (or RETURN) key, and a stop (.) will be printed at the beginning of a new line. This is your prompt to type your LOGON command. A stop character at the beginning of a line always indicates that the terminal is open for you to type a command.

If at any time you need to interrupt output to the terminal (for instance when you have seen enough of a file you are typing) you should hit CONTROL E twice. You will then receive the prompt character (.) and can type what you wish - probably HT (see 2.6 below). Hitting the BREAK key on a PACX terminal will disconnect you from VM.

If the connection is broken while you are using the PACX terminal you will be disconnected from CMS. To access your machine again you should reconnect your terminal to the 3032 and type your normal LOGON command again. Then type either BEGIN, or just B and continue your terminal session. You will then continue at exactly the point where you were disconnected from VM; if you were editing, you will still be in edit mode.

2.2.2 Access via a Workstation

There is currently no access available from workstations.

2.3 Logical Line Editing Symbols

Four logical line editing symbols are supplied to help with entering commands or lines of data from your terminal. The default character values for these are:

logical character delete	backspace (hex 16 - usually CONTROL H)
logical line end	#
logical line delete	\ (hex E0)
logical escape	"

2.3.1 Logical Character Delete (backspace, usually CONTROL H)

This allows you to delete one or more of the previous characters entered. Every character delete you type will delete 1 character (including the \ and # logical editing characters).

For instance:

XYZbackspace will give you	XY
XYbackspaceZ will give you	XZ
XYZ\backspacebackspace	will give you XY

2.3.2 Logical Line End (#)

This allows you to type more than one command, or line of data, on a line, for instance, typing

```
XYZ#ABC#DEF
```

will result in the lines

```
XYZ
ABC
DEF
```

being entered.

This character is also sometimes used when sending commands to CP in the form

```
#CP command
```

If you change your logical line end character for any reason you <u>must</u> use your <u>new</u> logical line end character in this command instead of #. For this reason it is important that you do not disable the logical line end symbol (by setting it to OFF) when using an ASCII terminal, otherwise you will remove the way of communicating with CP without going via the CMS system i.e. you are unable to use the #CP command. See chapter 6.

2.3.3 Logical Line Delete (\)

This deletes the entire line, or the last logical line up to and including the logical line end character (#). If a # immediately precedes a \ only the # will be cancelled. When a line is cancelled using this symbol there is no need to send the line to CMS before continuing - you can continue entering data on the same line.

For example:

```
XYZ#ABC\        gives you             XYZ
XYZ#\           gives you             XYZ
XYZ#ABC\#DEF    gives you the lines   XYZ
                                      DEF
ABC#XYZ\DEF     gives you             ABCDEF
XYZ\            deletes the whole line
```

2.3.4 Logical Escape (")

This causes the next character to be considered as a data character even if it is set up as a logical line editing symbol. For instance,

```
WXY"\Z          gives you          WXY\Z
""XYZ""         gives you          "XYZ"
```

Entering consecutive logical escape characters in conjunction with other logical line editing symbols can give unexpected results, for example

```
XYZ""backspaceABC and
XYZ""backspacebackspaceABC
```

both give

```
XYZABC
```

- if you enter a single logical escape symbol by itself, or as the last character on a line, then it is ignored.

You can find out what logical line editing symbols are set up at your terminal by using the CP command QUERY TERMINAL (see chapter 7). You can change a line-editing symbol by using the CP command TERMINAL (this is described in the CMS User's Guide).

2.4 Logging On and Off

When you have established contact with VM/370 (i.e. 'VM/370 ONLINE' has been displayed on your terminal), you can logon. Simply type

```
LOGON userid
```

where userid is your username. You will receive the reply

```
ENTER PASSWORD
```

and you reply by typing in your password. On most terminals this will be masked. VM replies by typing out any Log Messages and then

```
LOGON AT hh:mm:ss GMT day date     from CP
VM/370 - CMS RELEASE 6 PLC 4       from CMS
```

Alternatively you can type your password on the same line as your LOGON command, i.e.

```
LOGON userid password
```

and the Logon reply from CMS will be received immediately.

You are now ready to execute your PROFILE exec, and you do this by pressing the ENTER (or RETURN) key. (Your PROFILE EXEC is a file with filename PROFILE and filetype EXEC which contains commands you would usually send at the beginning of a terminal session). Normally a standard profile EXEC will have been set up for you. This will include a command to access the disk which contains new additions to the CMS system and, for ASCII terminals, a command to define the terminal page size (see section 2.6). The replies to the commands appear at the terminal followed by the standard CMS ready message:

```
R;
```

Whenever you receive this message it means that CMS is ready for you to send another CMS command. You should always wait for this message and not type ahead as this has the effect of putting you out of CMS mode and into CP mode, when all the commands are sent to CP. If this should happen by accident, type BEGIN (or B) to return to CMS. You can find out which mode you are in at any time by typing a null line, when you will receive the reply CP or CMS to indicate the mode.

If you do not press ENTER/RETURN immediately after logging-in, your PROFILE will be executed the first time you press the ENTER/RETURN key after typing in a command, with one exception. If you do not wish to run your PROFILE EXEC then the <u>first</u> command you type must be

```
ACCESS (NOPROF
```

To Logoff from CMS simply type the command

```
LOGOFF
```

VM will reply with two lines of information detailing your connect time, your virtual CPU time, your total CPU time and the date. If you were connected via the Gandalf PACX, the connection from your terminal to the 3032 will be dropped.

2.5 <u>Command Syntax</u>

In general commands to both CP and CMS are in the form

```
command op1 op2 ....... opn (opt1 opt2 opt3 ....... optn
```

where

command	is the command name, usually a verb
op1 to opn	are positional operands

opt1 to optn are mostly non-positional options

The command, the operands and the options are separated by one or more blanks - no commas are used. Blanks around the bracket separating the options from the operands are optional.

The length of all operands and options is limited to 8 characters; if more than 8 characters are typed CMS will truncate the string to the first 8.

Options are mostly non-positional, but in many cases you will find that one option is a keyword and the following option is its value.

For instance, consider the command NPRINT which is used to print CMS files under MVT. There are three operands - filename, filetype and filemode but if omitted filetype will default to LISTING and filemode to *, so you might type

```
NPRINT MYFILE (ID IDA ACCT ACNO CC
```

where your file is called MYFILE LISTING, you wish the job to run under the identifier IDA and the account number ACNO (so ID and ACCT are keyword options); CC is a non-keyword option specifying that the file contains carriage control characters.

2.6 Halting the Current Activity

The CMS immediate commands HT (halt typing) and HX (halt execution) are available for halting the current activity in your machine. If, for instance, you are typing a file at your terminal and do not wish to see any more of it you should gain access to your keyboard (on a PACX terminal by pressing CONTROL E twice as described in 2.2) and type

```
HT
```

This command can also be used to stop the output from a program running in your virtual machine being displayed at your terminal; the program will continue to execute and in this case the typing can be resumed by using the CMS immediate command

```
RT
```

If you have a program running in your virtual machine and you wish to terminate execution, type

```
HX
```

This will cause the program to terminate abnormally. Several lines of terminal output may be produced before the user is given control again.

On an ASCII terminal you can set a terminal pagesize by entering the CP command

```
TERMINAL PAGESIZE n
```

where "n" is the number of lines per page. This will cause replies from, for example, the TYPE command to stop every n lines. Entering anything or just a null line at this point will cause the output to continue. To halt the output you must press CONTROL E twice and enter HT as described above. (The TERMINAL PAGESIZE command will probably be executed within your standard PROFILE EXEC).

2.7 The HELP Facility

This facility enables the user to obtain help at the terminal with the use of CP and CMS commands and with Edit sub-commands. If you type

```
HELP command
```

you will receive information on the way to use the command specified by 'command'. For instance,

```
HELP TYPE
```

will tell you how to use the TYPE command.

If, instead of providing the command name in full, you use an abbreviation, like HELP T, you will be asked whether you want to continue with a search for the abbreviation - if you do, just press ENTER/RETURN; if not, type STOP.

If you wish to obtain a list of commands, type

```
HELP *
```

This will give you a list of all commands for which HELP information exists. Typing

```
HELP ?
```

will give you a list of commands, grouped by type, with a short description of each command.

To find out about Edit sub-commands type

```
HELP EDIT sub
```

where 'sub' is the sub-command name. With this command it is necessary to provide the name in full.

If you cannot remember the format of the HELP command itself, just type

```
HELP
```

and you will be told how to use this command.

If you want to use the HELP facility whilst using the Editor there are two macros available. To obtain information on an Edit sub-command type

```
$HELP command
```

where 'command' is the full name of the edit sub-command.

To obtain information on a CMS command type

```
$HELPCMS command
```

where again 'command' is the full name of the command; in neither case may an abbreviation be used at present.

Chapter 3: CMS FILING SYSTEM

Files in CMS are arranged in minidisks, each of which occupies all or part of a real disk. A CMS machine will normally have read-write access to one minidisk and read access to several others. Each minidisk is referenced by a letter for convenience and conventions exist for assigning letters to particular disks.

A	This will normally be the user's own read-write disk where his own files are kept.
S	This disk is where the CMS system and many of the CMS standard commands reside.
Y	This is an extension of the standard system disk and contains CMS commands added by RL.

The minidisks above are accessed automatically for the user at IPL time. Other disks may be accessed as required after login, provided the necessary passwords are specified. Another disk containing new software provided by RL, is often accessed as the G disk (see NEWSOFT command). The default PROFILE EXEC supplied to new users ensures that the user has access to the NEWSOFT minidisk.

By linking and accessing another user's A-disk, you can gain read-only access to his files. An EXEC called RLINK provides this for you (see later in this chapter).

3.1 CMS File Identifiers

Each CMS file is referenced by an identifier which has three parts, the filename, filetype and filemode. Some commands such as COPYFILE and RENAME require the whole identifier to be specified, whereas others permit defaults or assume particular filetypes or filemodes.

Filename	This is an alphanumeric name of up to 8 characters which is chosen by the user. It can be all numeric.
Filetype	This field is also alphanumeric, up to 8 characters and is chosen by the user. However it should be noted that several utility programs such as compilers, assemblers and editors, make use of this field. For instance the Fortran compiler will expect input files to have filetype FORTRAN and the editors set up many defaults (such as record format, tab settings and translation of text) according to filetype.

Some common reserved filetypes are:-

FORTRAN -	used by FORTRAN compilers
PLIOPT -	used by PL/1 compilers
ASSEMBLE -	used by Assembler
LISTING -	listing file produced by compilers with carriage control
TEXT -	object decks produced by compilers
EXEC -	file of executable commands.

Filemode This field is 2 characters, made up of a letter and a number (0 to 5). The letter indicates the disk on which the file resides and the number controls special attributes. The number indicates certain characteristics of the file such as making certain files private, temporary or specifying OS format for new files. These are described fully in the CMS Users Guide, section 4. In many commands it is not necessary to give the filemode, the default being A1.

3.2 Creation of Files

The simplest way to create a file is to use the Editor e.g.

```
EDIT FRED FORTRAN
```

If the file does not exist, the editor warns you, and puts you in input mode until you type a null line (just a return). During INPUT mode the editing inserts the lines you type after the current line. After the null line, you remain in the editor and the file is not written to disk until you type "FILE", which terminates the editor (see section below "Editing Files").

By default, the editor sets many file characteristics according to the filetype. For instance by default a file named FRED FORTRAN A1 will have fixed length records, 80 characters long and the editor will truncate lines longer than 72. All these settings, apart from the maximum record length, can be changed after entering the editor. See the RL VM Manual section on the "Extended VM/370 Editor".

3.3 Editing Files

The Extended CMS Editor provides a powerful set of subcommands for display and modification of CMS files. Lines up to 255 characters long can be displayed (in EBCDIC or HEX) or altered and changes are verified immediately at the terminal. Up to 9 files can be edited simultaneously by issuing the EDIT command within the editor.

Details of the extensions are given in the RL VM Manual section on the "Extended VM/370 Editor". A full user's guide for the editor is circulated with the RL VM manual. A subset of the facilities is described briefly below. The capital letters in the subcommand name indicate the minimum abbreviation recognised. In commands where a string may be specified, without a trailing delimiter, trailing blanks in the command line <u>are</u> significant. In other cases, such as between a command and its operands, the space is normally optional, so that "DOWN 5" and "D5" are equivalent.

In commands where searching takes place, only the part of the file within the current zone settings is used. Suitable defaults are set for the zone by the editor according to filetype and the setting may be changed while in the editor by using the ZONE command.

During an editing session, the file is kept in virtual storage. Consequently a system crash would cause all the modifications to be lost. This can be avoided by making use of the SAVE or AUTOSAVE sub-commands as described below.

In the following description of some of the more useful edit sub-commands, the < and > brackets are used to indicate optional operands and the minimum abbreviation for each sub-command is given in upper case.

3.3.1 Input <line>

If the "line" is provided, this command inserts the given text as a new line after the current line. If the "line" is not provided, the user is placed in input mode (expecting one or more lines of input) which is only terminated by a null line (return). At least one space must separate the end of the command from the line itself, further spaces are considered to be part of the line. The Replace command acts in the same way as the Input command, except that the current line is deleted.

3.3.2 Type < n >

This command displays 1 or more lines starting at the current line. The default for n, the number of lines, is 1: * indicates that the display should continue to the end of the file.

3.3.3 TOp

This moves the current line pointer to be just above the first line of the file.

3.3.4 Bottom

This moves the current line pointer to be just below the last line of the file.

3.3.5 Find string

This command can be used to locate a line beginning with the given string. Starting after the current line, each line is examined from the beginning, testing only the characters corresponding in position to the non-blank characters in "string". When all the non-blank characters of the string are found in the same positions in the line, the search terminates and the new current line is displayed. If the search is unsuccessful, the current line pointer is positioned at the end of the file. At least one space must follow the command, further spaces being considered part of the string.

3.3.6 Locate /string/

This command can be used to search for "string" from the current line onwards, anywhere within the current zone settings. The "/" indicates any non-blank delimiter (not part of the string) except "." which has special uses (see PERIOD command in the editor manual). If found, the new current line is displayed, otherwise the current line pointer is positioned at the end of the file. The second delimiter is optional and if omitted, trailing blanks are treated as part of the string.

3.3.7 Down < n > or Down /string/

This command repositions the current line pointer according to the sort of argument given. If "n" is specified, the pointer is moved n lines down (default is 1 line down). If the string is specified, the new current line becomes the first line containing the string within the current zone settings.

3.3.8 Up < n > or Up /string/

This command behaves the same way as DOWN except that the movement and searching are backwards.

3.3.9 Change /string1/string2 < / < n < * >>>

This command exchanges string1 for string2 on one or more lines anywhere within the current zone settings. "n" is the number of lines (starting at the current line, default 1) to be searched and "*" directs the editor to change all occurrences. The default is only to change the first occurrence on each line. Each changed line will be verified at the terminal.

e.g.

```
C/FRED/JIM/1 *
```

will change <u>all</u> occurrences of FRED to JIM in the current line.

```
C/FRED/JIM/* *
```

will change all occurrences of FRED to JIM in the current line and through to the end of the file.

3.3.10 APPend string

This command appends the given string to the end of the current line. If there is more than one blank following the command it is considered to be part of the string. The appending takes place only within the current zone settings.

3.3.11 DElete < n > or DElete /string/

This command deletes "n" lines (default 1) beginning with the current line. If "n" is negative, n lines will be removed, the last one being the current line. If the string argument is used, the last line deleted will be that containing the first occurrence of the specified string (within the current zone setting).

3.3.12 FILe < fn < ft < fm >>>

This command terminates the editor and writes the file to disk. The default is to overwrite the original file although the arguments may be used to specify a new file identifier. In this case the original file will be unchanged. N.B. If a file of the new name already exists, it will be replaced without warning.

3.3.13 SAVe <fn <ft <fm>>>

The SAVE command is identical to the FILE command (see section 3.3.12) except that the editor is not terminated. The current copy of the file is written to disk, overwriting the original file by default. It can be used periodically during a long editing session to avoid losing all modifications during a system crash.

3.3.14 AUTOsave <n>

This command instructs the editor to checkpoint the current state of your file every time n lines are changed, deleted or inserted. If n is not specified, the current setting is displayed. The current copy of the file is written to a temporary file so that a subsequent QUIT command retains the original version of the file (as it was originally or after the last SAVE command). The temporary file has the filetype and filemode of the original file and a filename of the form $$hhmmss where hhmmss is the time of writing the file. In the event of a connection break or system crash, the latest of these files will remain. It is deleted when the editor terminates normally by FILE or QUIT commands.

3.3.15 QUit

This command causes termination of the editor immediately without writing anything to disk. The original file remains unaltered unless the SAVE command has been used (see section 3.3.13).

3.3.16 Again <n>

This command repeats the last command "n" times (default 1).

3.4 LISTFILE

This command provides a convenient method of displaying information about a specific set of the accessible CMS files. The output can be to terminal, disk file (CMS EXEC A1) or printer. The information available can include name, record format, size and date last written and the files may be selected by name, creation date, record format, size etc. The resulting list may be sorted in a variety of ways before output. The command is fully described in the CMS command section of the RL VM Manual. Some common uses are shown below.

```
LISTFILE
```

This will list at the terminal all files on your A-disk, showing the full name. The names are placed three to a line.

```
LIST * FORTRAN
```

This will provide a similar list of all the files on the A-disk with filetype FORTRAN.

The CMS filing system does not contain a directory structure. A logical directory structure could be imposed by using the first two characters of the filename to group files together. The LISTFILE command could then be used to list all files with the same first two characters in the filename.

```
L XY*
```

This will list all files with filenames beginning XY.

```
L DA* ASSEMBLE (E
```

This will write a file called CMS EXEC A1 which will be an EXEC file containing the names of the files on the A-disk whose filenames begin with characters "DA" and are of filetype ASSEMBLE. See CMS Users Guide, section 6, for details of EXEC files.

3.5 COPYFILE

This is an extremely powerful command which can be used to copy and/or modify some or all of the files on a CMS disk. Particular records and specific columns of records may be copied or overlayed onto those of an existing file. Record formats may be changed and translation to upper or lower case (or specific translations) may be requested with the other facilities. Full details are given in the CMS Command and Macro Reference. Some simple uses are shown below.

```
COPY FRED FORTRAN A1 JIM JOB A1
```

This will make a copy of the first file in JIM JOB A1 which will have the same record format and length.

```
        COPY FTYPE * A1 FTYPEO  = = (REP
```

This will copy <u>all</u> the files on the A-disk with filename FTYPE and <u>overwrite</u> existing files with filename FTYPEO but retaining the original filetypes and the filemode. The REP option permits replacement of existing files and the "=" sign retains the corresponding value of the input argument.

3.6 <u>RENAME</u>

This command permits renaming of existing files on a CMS disk. All parts of the name except the filemode letter can be changed. The new name of the file should not conflict with existing files.

```
        RENAME FRID FORTRAN A1 FRED = =
```

As with the COPYFILE command, the "=" sign can be used to retain the old values of the filetype and filemode.

3.7 <u>ERASE</u>

This command is used to delete existing files from your read-write disks.

```
        ERASE FRED FORTRAN
```

will delete file FRED FORTRAN A1. The "*" facility may replace the filename or filetype (but not part of either as in LISTFILE).

```
        ERASE * LISTING
```

will delete all files of filetype LISTING on you A-disk.

3.8 <u>RLINK</u>

The command RLINK, which is a locally written EXEC, can be used to link to and access another user's A-disk. The format of the command is

```
RLINK  userid  password  letter
```

where "userid" is the user's identifier, "password" is the read password for the user's A-disk and "letter" is the filemode letter to be used when referring to files on this disk.

```
e.g.  RLINK  FRED  RFRED  B
```

will link to user FRED's A-disk (as long as it has read password RFRED) and access it as filemode B.

RLINK gives read only access to another user's A-disk. It is strongly recommended that users do not obtain write access to other users' disks as there is no protection against simultaneous updating of a user's directory.

When the link is no longer required, the disk may be released and detached by using the RELEASE command with the DET option.

```
e.g.  REL  B(DET
```

All links to other disks are automatically removed on logging out.

For further information on linking to, accessing and releasing other minidisks see the CMS User's Guide, Section 1, pp.13-15.

3.9 MVTDISK

This is an RL written EXEC to provide convenient access to the OS disks used by the MVT batch systems. It performs the LINK and ACCESS commands for you.

```
MVTDISK ?
```

will type out a list of the disks available.

```
MVTDISK FREEDISK E
```

will access FREEDISK as your "E" disk (any disk previously accessed as "E" will be released). Type HELP MVTDISK for details of this command.

Once this connection has been made, it is possible to read sequential datasets into CMS (and members of partitioned datasets). The RL written EXEC "OSCOPY" has been provided for this purpose. You can also read these datasets from programs running in your CMS machine, by issuing FILEDEF commands which are the CMS equivalent of DD statements. See CMS Command and Macro Reference for full details of the FILEDEF command.

3.10 OSCOPY

This RL written EXEC has been provided to run a program in your CMS machine to copy sequential OS datasets or members of partitioned datasets into CMS files.

```
        OSCOPY fn ft fm <osfm < dsn >>   < ( options >
```

The OS dataset name "dsn" must be given as separate concatenations with the normal periods replaced by blanks. The options permit specification of member names of partitioned datasets and allow appending to, or replacement of existing CMS files. Type HELP OSCOPY for details on how to operate the EXEC.

```
        OSCOPY FRED FORTRAN A1 E JAN LRMACRO
```

This command will create CMS file "FRED FORTRAN A1" (provided it did not already exist) by copying OS dataset JAN.LRMACRO from the disk accessed as the "E" disk (see MVTDISK command above).

Chapter 4: PLANT AND SUPPLY SYSTEM

4.1 Introduction

The Plant/Supply program has been transferred from the 4080 Data Editing System in order to provide improved file concatenation and dynamic parameter substitution in CMS. It is anticipated that more comprehensive facilities will be provided later.

4.2 General Description

This facility is modelled on the $ADD, $SUPPLY and $PLANT concepts in the Rutherford multi-access system, ELECTRIC, which runs under HASP and OS/MVT.

It enables users both to combine files and substitute text dynamically at the time when a file is used. Special declarations within the file, introduced by a delimiter character, are replaced by either a text string or a file. The declarations contain names to link them to parameters supplied by the user when the program is initiated. Each declaration may also contain a default value for the text string or filename to be applied when the user supplies no parameter of his own.

Substitutions are introduced by the delimiter '{' and terminated by '}', although these may be altered.

Spaces may appear within the substitution declarations to separate various items of the declaration from each other. However care must be taken with the plant form when supplying a default text string to differentiate between a null string, a string containing spaces and no default string. Conceptually, newlines in a text file may be considered to occur at the end of each line of the file.

All substitution definitions may appear within lines or on lines by themselves. If a change of delimiter or global declaration appears on a line by itself then this line is suppressed on output.

Users should ensure that inclusion of text strings and files does not result in lines longer than 132 characters being output.

In the following format descriptions < and > are used to denote optional items.

4.3 Text Insertion ($Plant Form)

Text insertion is achieved by including a string of the following form anywhere within a line.

{ <name> : <text> }

name	This may be any alphanumeric name up to a maximum of 8 characters in length. Although 'name' is optional in the plant form, the substitution specification is pointless without it since no user parameter can be given.
text	This is a text string which acts as a default value if the user supplies no parameters of his own. The text string need not be enclosed in quotes unless:-

(i) A null string is required when a pair of quotes ("") is used.

(ii) The string begins with a space since leading spaces are ignored.

(iii) The text begins with the literal *.

The text string may not span lines. If 'text' is specified as * then the global default for this name is applied (see GLOBAL DEFAULT DECLARATIONS).

Text insertion definitions may not be nested although the text string itself may contain further substitutions (both text and file). Nesting of the text string to a depth of 5 only is allowed.

4.4 File Insertion ($Supply/$Add Form)

File insertion is achieved by including a string of the following form anywhere within the file.

{ <name> = <filename> }

name	This is the same as for the plant form. To achieve the effect of the add form when the file to be inserted is predefined and does not need to be dynamically supplied, the name is not required.
filename	This is any valid CMS fileid, or OS ddname. It acts as the default file to be inserted if no user supplied parameters are found. If filename is specified as * then the global default for this name is applied (see GLOBAL DEFAULT DECLARATIONS). If an OS ddname is given, you must have previously issued a FILEDEF command to define the OS dataset required. The program terminates at this point if no FILEDEF was issued. Note that the PLANT EXEC provides a FILEDEF for ddname DUMMY (unless you

have already set one up) in order to permit a default of an empty file in the supply declaration. See the example below.

A file which is being substituted may itself contain further definitions of text strings or of files. Nesting of files to a depth of 5 is allowed. The file insertion definitions themselves may not be nested although the local default 'filename' may contain further substitutions of type text only. Nesting of substitutions of type file within 'filename' are not allowed.

4.5 Global Default Declarations

Global default declarations may be set by including a string of the following form anywhere within the input file.

{ name ; string }

Global default declarations serve as default text strings or filenames for use when a text or file substitution specifies a local default value of *.

Global default declarations act for all substitution definitions, both file and text, with the same name. They are valid only for the input file in which they were originally declared.

Whether the global string is to be used as a filename or text string is not defined within the declaration itself but only in the type of substitution specifying the <u>use</u> of the global default. Therefore nesting of the global string must follow the same rules as for nesting of text strings or filenames, depending on the context in which it is to be used.

Change of delimiter declarations or further global declarations may not be nested within a global declaration.

A further restriction on global strings is that if substitutions are nested within the string then these substitutions must not declare a local default of *. Similarly the global default string itself must not be *.

Any nested substitutions occurring within the global string are not resolved until the global is used. However checks are made on the syntax and validity of the string. If the string contains nested substitutions, a check is made to ensure that either the substitution name occurs in the user argument list or that a local default (not *) is present. If this check fails then processing terminates since the global declaration is unresolvable.

4.6 Change of Delimiter Declarations

The delimiters, which are recognised by the PLANT program, may be changed by including a string of the following form anywhere within the input file.

{ a ! b }

The current substitution delimiters which default to { and } are changed to a and b respectively. The change of delimiter remains in force until either a new change declaration is processed or until a new file is inserted. Then the delimiters revert back to their default values i.e. the change of delimiter will only act on the file in which the change was made.

The restrictions on delimiters are that neither a nor b may have the values ';', '!', ':', '=', '*' or space. If a is set equal to b then further processing of substitutions ceases and all input is copied directly to the output stream.

Change of delimiter declarations may not be nested.

4.7 Operation - User Interface

At the start of processing the program asks the user for any parameter settings with the prompt:-

USER ARGUMENTS>

This is then the opportunity for the user to override default values of text strings and filenames given in substitution definitions. The syntax of the reply given is a series of name=value expressions, separated by commas and terminated by a semi-colon or a null line. The parameter list may be continued on to a new line provided that the continuation occurs after a value and before the next parameter name. If a null value of the parameter is to be given then a comma must follow directly after the equals sign. Strings may be enclosed in double quotes if the string is to contain either of the special characters ',' or ';'. To obtain the double quote character within a string enclosed in quotes, the character should be doubled up. If no arguments are to be supplied then the user should press carriage return.

During processing, any parameter whose value is needed (i.e. has no default supplied) but which has not been given by the user, is prompted for at the terminal. The type of value required (i.e. filename or text string) is indicated and the name of the file containing the substitution declaration is given. At this point the user should supply the value only and terminate it by newline.

On completion of processing, the message "PLANT COPY COMPLETE" is output to the terminal.

4.7.1 Running the Program

The program is operated by a CMS EXEC of name PLANT. The command takes one of two possible forms:

1 Output to terminal or to VM spool, for printing, punching, or job submission.

```
    PLAnt fn <ft <fm>>  < ( <|Submit|> <Noargs> <Mixed> >
                             |Term  |
                             |Print |
                             |PUnch |

 defaults:    JOB   A1          Submit
```

2 Copy to another CMS file.

```
   PLAnt fn1 ft1 fm1 fn2 <ft2 <fm2>> <|( <|App|> <Noargs> <Mixed> >
                                     |    |Rep|
                                     |
                                     |  <RECfm |F|> <Lrecl n> <Noargs>
                                     |         |V|        <Mixed>

 defaults:                  ft1  fm1       Recfm F     Lrecl 80
```

In both forms the defaults for input filetype and filemode are JOB A1. The defaults for ft1, fm1, ft2 and fm2 may be specified by giving "=" as the argument. The default filetype and filemode for the second form are the input values of these parameters. The input CMS file can have fixed or variable length records and by default a newly created file will have fixed length records, 80 characters long. When appending to an existing file the original record format will be retained. The record length is retained if RECFM=F and lines longer than this will be truncated. If RECFM=V, truncations only occur after 80 characters or the previous maximum LRECL, whichever is larger.

The options provide the following facilities:

Noargs	Used with either form to prevent the prompt for user arguments at the initiation of the program.
Mixed	This option prevents translation of terminal input to upper case. By default no translation occurs when the input filetype is MEMO, LAYOUT, GEROFF or SCRIPT.
Submit	This option spools the output to the MVT Batch System. The virtual machine punch is routed to the MVT system to produce the output and is set back to the VM spool afterwards. This is the default option.

Term	Output is displayed on the terminal.
Print	Output is printed to the VM Spool. The routing of the virtual machine printer is not changed.
PUnch	Output is punched into the VM Spool. The routing of the virtual machine punch is not changed.
App	The output file is to be appended to an existing CMS file.
Rep	The output file is to replace an existing CMS file.
	Error messages are produced if the output file already exists, unless one of the options APP or REP is used. A warning message is produced if the output file does not exist when the options APP or REP are used.
RECfm c	The record format for a newly created CMS file is to be 'c'. Only values 'F' or 'V' will be accepted. This parameter is ignored when the Append option is used.
Lrecl n	The record length of a newly created CMS file is to be 'n', which must be numeric. This parameter is ignored if the Append option is used.

4.8 Errors and Error Recovery

Various errors may occur during processing. A list of these is given at the end of this section. In the case of fatal errors the message "PROCESSING TERMINATED" is output to the terminal. For non-fatal errors the user is given the option of continuing processing. All error messages are prefixed by the name of the file in which the error occurred. In the case of fatal errors a condition code of 8 is returned and for other errors a condition code of 2. If the user supplies an argument which is not used by PLANT, then a condition code of 4 is returned with a warning indicating the name of the unused argument. This error is not fatal and PLANT completes normally.

If illegal nesting within a substitution definition is detected then the error recovery procedure is to match delimiters and ignore the whole substitution. This action is also taken if the type of substitution (file or text) is not indicated. For illegal nesting within a filename only the illegal substitution is ignored and an attempt is made to process the substitution containing it.

4.9 Examples

This example shows both forms of the PLANT command operated on the file below. The file contains simple MVT JCL to compile (with G or H compiler) a small Fortran program and run it. The Fortran source is assumed to be in CMS file TEST FORTRAN, some linkedit JCL in TEST LINKEDIT and the data in TEST DATA. The first line changes the delimiters from "{" and "}", to "<" and ">". Note that the second supply <ROUTINE=DUMMY> has no effect unless specified in the user arguments, since the EXEC provides a dummy FILEDEF for ddname DUMMY.

```
{<!>}
/*PRIORITY <PRI:8>
//IDJOB JOB (<ACCT:4460>,<ID:LR>),'*'
// EXEC FORT<COMPILER:>
<SOURCE=TEST FORTRAN>
<ROUTINE=DUMMY>
<=TEST LINKEDIT>
//G.SYSIN DD *
<DATA=>
/*
```

The following is an example of using the PLANT command to copy to another CMS file. The option "n" prevents the initial prompt for user arguments. Since no defaults were given for "COMPILER" and "DATA", the program prompts for their values.

```
plant test job a test output(n
PLANT VERSION 2.0
FILE 'TEST     JOB      A ',TEXTNAME 'COMPILER'>
g
    FILE 'TEST     JOB      A ',FILENAME 'DATA'>
test data
    PLANT COPY COMPLETE
R;
```

Below is the newly created file TEST OUTPUT.

```
/*PRIORITY 8
//IDJOB JOB (4460,LR),'*'
// EXEC FORTG
C A SIMPLE FORTRAN PROGRAM
      STOP
      END
//L.LIB DD DSN=ULIB.CMS,DISP=SHR
    INCLUDE LIB(LR)
//G.SYSIN DD *
THIS IS MY DATA
/*
R;
```

This example shows this file now being submitted to the MVT batch system using the other form of the PLANT command. User parameters are specified to override the defaults for PRI and ID.

```
plant test
PLANT VERSION 2.0
USER ARGUMENTS>
pri=12,id=lr

FILE 'TEST     JOB      A1',TEXTNAME 'COMPILER'>
g
     FILE 'TEST     JOB      A1',FILENAME 'DATA'>
test data
     PLANT COPY COMPLETE
PUN FILE 6106  TO  CEM      COPY 01 NOHOLD
DMSPLA006I JOB SENT TO MVT SYSTEM
R;
```

Chapter 5: RUNNING PROGRAMS

5.1 Introduction

There are three ways in which a program can be run. It can be submitted to the MVT system as a batch job, it can be run under CMS in the user's machine or it can be submitted to CMS batch. This chapter provides enough information to run simple programs and provides guidance on the most suitable environment for a particular program.

5.2 MVT Batch Execution

5.2.1 Job Submission

A job can be submitted to the MVT batch system by use of the SUBMIT command. For example

```
SUBMIT FRED PROG A
```

This submits the file with the filename FRED of type PROG on the A-disk to the MVT system. There is a default filetype of JOB and filemode of A so by using JOB as a filetype for MVT bound work the command can be simplified to

```
SUBMIT BILL
```

which is equivalent to

```
SUBMIT BILL JOB A
```

The file must include all the necessary JCL etc. to satisfy MVT in just the same way as if it were submitted from a real card reader. The user should consult the RL CIGAR manual for details of these facilities.

5.3 Output Retrieval

Currently this can only be achieved through the normal MVT output routes, e.g., on a lineprinter at a workstation. There are plans to enable output to be routed to a CMS retrieval machine from which it could be viewed at a terminal. It would remain in the retrieval machine for a week or two after which it would be deleted. For the longer term retention of output the user will be expected to copy it to his own minidisk. The retrieval machine will handle both text and graphics output. This section will be updated as further details become available.

5.4 Job Status

The path of a job through the system to and from MVT is complex. To ensure that the user has access to adequate information about his job it is planned to run a status virtual machine under VM which will be kept informed by the operating system of changes in a job's status as it progresses through the system. There will be simple commands available to interrogate this machine and provide the requested information at the terminal. This section will be updated as work in this area progresses.

5.5 CMS Interactive Execution

5.5.1 Introduction

Programs can be run under CMS in two ways. The user can use his own virtual machine for compilation, execution, etc. or he can submit a job to a CMS batch virtual machine. In the latter case output is spooled to the user's virtual reader after the job completes. This has the advantage that the user can carry out other tasks while the job is running and he has access to more store (up to 2M) in the batch machine.

5.5.2 Compilation and Assembly

A job is compiled in the user's machine simply by stating the name of the language processor followed by the filename. For example

```
FORTGI  BERT
```

compiles a file BERT FORTRAN with the Fortran G1 compiler. The filetype must be FORTRAN; it cannot be specified on the command. A search is made, in search order, for the file on all disks accessed by the user's machine, the file usually residing on the A-disk.

For other languages similar rules apply. PLIOPT invokes the PL/I optimising compiler, expecting files of type PLIOPT. FORTHX is available for the FORTRAN H EXTENDED compiler but the CMS version cannot be overlayed and so takes more than the 512K maximum machine allowed to users. It must therefore be sent to CMS batch. A filetype of FORTRAN is expected as for the G compiler. In fact PLIOPT is best run in batch for large programs because it expands to use the space available and takes much longer if given too little storage. Other languages are also available as described in the RL VM Users Reference Manual.

A successful compilation creates two files with the same filename as the compiler source file. Compilation of BERT produces BERT LISTING and BERT TEXT. The LISTING file contains the equivalent of the line printer listing while the TEXT file contains the object deck, i.e. the compiled program. If the compilation is unsuccessful the TEXT file is not created and the return code is non-zero. Previously generated LISTING and TEXT files of the specified filename are overwritten.

5.6 Running a program under CMS

There are a number of ways of executing a job in CMS. Starting with a TEXT file generated from a FORTRAN compilation it is first necessary to issue a GLOBAL command to allow access to the Fortran Library. It is often convenient for a user to add this command to his PROFILE EXEC if he uses Fortran most of the time. The LOAD command is used to load one or more TEXT files into virtual storage. At this point CMS attempts to resolve external references by searching firstly for TEXT files with the relevant names and secondly for library routines in any TXTLIB's previously made accessible through the GLOBAL command. The program can then be executed with a START command. If BERT was a stand alone program it could be run thus

```
GLOBAL TXTLIB FORTLIB
LOAD BERT
START
```

There are alternative means of issuing LOAD and START as one command either by

```
LOAD BERT ( START
```

or alternatively by

```
RUN BERT
```

There is also a means of producing a load module. Consider the following

```
LOAD BERT
GENMOD BERT
BERT
```

In this case a file BERT MODULE is created by the GENMOD command and is executed by simply issuing the filename, BERT. The load map is stored in a file called LOAD MAP. Note that if the filename is not specified with the GENMOD command, the load module name will be that of the first entry point in the load map, usually MAIN when using FORTRAN. Some care should be taken in the choice of file names because any file of type EXEC which is accessible will be executed in preference to a MODULE of the same name.

If the program needs to read or write CMS files, the user will need to issue FILEDEF commands which are described in the CMS User's Guide. In Fortran, streams 5 and 6 are linked to the terminal for input and output by default.

5.7 CMS BATCH - Introduction

CMS batch jobs run in a CMS batch virtual machine. This is like a user's CMS machine in most respects but is modified so that it tests for input on its virtual card reader and if not otherwise occupied reads files spooled to it. The file is then executed as a CMS job, subsequent files being queued on the batch machine's virtual reader until the current job has completed execution. Thus CMS batch expects to receive executable commands in much the same way as those entered at a terminal. Since the commands are executed in the batch virtual machine access must be provided to user disks as necessary and output will only be sent back to the user if he issues the necessary SPOOL commands. The user should note that CP commands must be preceded by "CP". The more advanced features available in EXEC files are often not necessary when submitting a job to CMS batch but the reader should consult the CMS Users Guide if more information is required.

5.7.1 CMS BATCH EXEC

To assist in the use of the batch machine there is an RL EXEC called BATCH. To submit a file for compilation, the command can be used in several ways. The simplest of these is of the form

```
BATCH fn ft
```

where fn is the filename and ft is a filetype recognised as that associated with a language processor. Those recognised at present are FORTRAN, PLIOPT, BCPL and ASSEMBLE. For example

```
BATCH JACK FORTRAN
```

will invoke the Fortran H EXT compiler (FORTHX). The BATCH EXEC builds a file suitable for submission to the batch machine. It creates a batch jobcard for CMS accounting purposes and by default will precede the source to be compiled by the necessary CP commands to ensure output is spooled back to the user. Assuming the file to be on the user's disk, it creates a LINK command after asking the user to supply the read password. This is then spooled to the CMSBATCH machine before the source file.

There are other forms of the command where the language processor can be defined explicitly. For example

```
BATCH fn ft (langproc

BATCH fn (langproc

BATCH fn ft langproc
```

are allowed so that

```
BATCH JACK FORTRAN (FORTGI

BATCH JACK (FORTGI

BATCH JACK FORTRAN FORTGI
```

all lead to a compilation using the Fortran G compiler.

It should be noted that specification of the filename and filetype, as in the simplest example, is unambiguous for all languages except Fortran and in most cases the simple form will be suitable. It is only when the user requires to perform a Fortran G compilation that a more complex form is necessary and in most cases the user will find that G compilations are ideally suited to run in his own machine and BATCH is not required.

The alternative mode of operation is to specify JOB as the third argument. Thus

```
BATCH JOHN MYJOB JOB
```

punches the file JOHN MYJOB from one of the user's accessed disks to the CMSBATCH machine. Spooling arrangements are as before but LINK, ACCESS, GLOBAL commands, etc., which are needed by the job when it runs in the BATCH machine, must be provided in the file.

If JOB is omitted, the file will still be submitted as a job to the CMSBATCH machine so long as the filetype (MYJOB in the example) is not recognised by the BATCH EXEC to be associated with a language processor.

There are also facilities for a file to be included just before and/or after the requested function (eg a compilation). This allows the user to issue more complex commands to set up the BATCH machine before the compilation, assembly or job, is initiated. Thus LINK commands to disks other than the user's disk can be issued as well as GLOBAL commands for

MACLIB's and TXTLIB's, while keeping the default and probably most used forms of the command simple. A complete description of these facilities is contained in the RL VM User's Reference Manual.

5.7.2 CMS BATCH - Output Retrieval

The Batch job is built so that by default it spools all output back to the user. In the case of a compilation, three files are punched to the user's virtual card reader, containing the same information as would have been created if the compilation had taken place in the user's machine. The first, a PRINTER FILE, has no header and is equivalent to the file of type LISTING produced from a compilation in the user's own machine. Therefore, it could be read into a file thus:

```
READCARD JOE LISTING
```

producing a file of type LISTING on the A-disk. The READCARD command will strip the file of its carriage control; the carriage control can be preserved by using the READPRT command instead:

```
READPRT     JOE
```

The second file is a PUNCH file and is the object deck - a TEXT file in CMS terms. It has a header and can be read simply by

```
READCARD *
```

producing a file JOE TEXT from JOE FORTRAN if the compiler input had a filename of JOE. The third file, a CONSOLE file, has no header, and contains the spooled console output from the batch machine. It therefore contains all the CP SPOOL commands and the compilation commands to CMS. It might be read into a file of type CONS by

```
READCARD JOE CONS
```

The user is reminded that he can reorder his files as explained in Section 7 of the CMS User's Guide (ORDER command) and read them in a different sequence. Note that only PUN files have the option of having a header; this is a feature of CMS but if a file is read by READCARD * assuming a header which is not there, the system will generate the name READCARD CMSUT1 A1 for the file so that it is not lost.

5.8 Where to run a particular job

There are no major restrictions on the type of work that can be submitted to MVT batch. The user should remember though that there is no access from an MVT job to files on CMS minidisks. In a complex job use should be made of User Libraries for storing compiled modules by submitting the source from CMS to MVT for compilation and then entering the module as a member of a library. Data held in CMS should pass in-stream with the job at submission time. Details of all these functions are described in CIGAR.

CMS is designed to run all types of work but some things are more easily done in MVT. Furthermore local restrictions, some imposed to ensure that CMS response is good and some because local software needs to be developed, make some jobs unsuitable for CMS. This situation may change as experience is gained in running a CMS service.

Compilation using FORTGI, PLIOPT and also assemblies are ideal to run in the user's CMS machine but FORTHX must be sent to CMS batch because it needs more than the maximum 512K allowed to users. Large compilations or assemblies might be slow and better sent to CMS batch leaving the user free to carry on with other work. Initially running jobs under CMS will be severely restricted until access to public libraries is provided and there will be no access to the MVT libraries such as ULIBs. Data on MVT disks can be read from CMS but not written.

There will be no access to tapes from CMS mainly because no interface exists between SETUP/TDMS and CMS.

To summarise then, initially CMS is best used for two main purposes. It is ideal for compilation both in the user's own machine or in CMS batch. It is useful for short test runs either accessing samples of data on OS disks or copies of data in CMS. The more powerful interactive debugging aids should greatly assist program development and the number of very short jobs which impose high overheads under MVT, should be reduced. There are few occasions where CMS should be used for production work and in most cases the user will find it impractical to do so.

Chapter 6: ERROR RECOVERY

This chapter provides some guidance on how to recover when you get your CMS machine into a mess.

6.1 Terminating Terminal Output

The CMS immediate commands HT (halt typing) and HX (halt execution) are available for halting the current activity in your machine. If, for instance, you are typing a file at your terminal and do not wish to see any more of it you should gain access to your keyboard (on a PACX terminal by hitting CONTROL E twice as described in 2.2) and type

```
HT
```

This command can also be used to stop the output from a program running in your virtual machine being displayed at your terminal; the program will continue to execute and in this case the typing can be resumed by using the CMS immediate command

```
RT
```

6.2 Terminating Execution of an EXEC or a Program

If you wish to terminate execution of an EXEC or a program running in your CMS machine, type

```
HX
```

This will cause it to terminate abnormally, though several lines of output may be produced at the terminal before the user is given control again.

Note that if your program or EXEC is actually reading from the terminal when you type HX or HT, the normal function of these commands will not be obeyed (unless you have specifically planned for this in your program or EXEC). In this case, it is first necessary to type

```
#CP EXT
```

which asks CP to send an external interrupt to your machine and thus put you into CMS DEBUG mode. The messages

```
DEBUG ENTERED
EXTERNAL INTERRUPT

DEBUG
```

appear and then the HX command will be accepted. Note that this facility uses the logical line end symbol and users are advised not to set this symbol OFF, or to set LINEDIT OFF while using an ASCII terminal as the facility becomes unavailable. On full screen terminals the PA1 key is designed to take you directly into CP mode from which you can type

```
EXT
```

followed by HX.

Following the HX command the message

```
CMS
```

appears to indicate that control has been returned to your CMS system.

6.3 <u>If all else fails</u>

If nothing seems to work at all it is possible that your own CMS virtual machine has been overwritten. In this case try IPLing your machine (initial program load) by typing:

```
#CP I CMS
```

which should return you to the login sequence (see section 2.4) just after the LOGIN command has been typed.

Note that most CMS commands are not recognised when typed in CP mode. On an ASCII terminal you can check which mode you are in by typing a null line. The message

```
CMS
```

or

```
CP
```

should appear accordingly. On a full screen terminal, the status indicator in the bottom right hand corner, shows the status of the machine.

Chapter 7: OTHER USEFUL FACILITIES

7.1 Message and Mail Facilities

A special command, RMAIL, has been set up to allow you to send messages and to mail files to other users. The messages and mail files are stored by a special virtual machine called MAILMAN so that the recipient does not have to be logged in when you send them. Messages are stored in a special file referred to as a MESSBOX.

7.1.1 Sending a Message

Type

```
RMAIL userid
```

where userid is the username of the person to whom you are sending a message. You will receive the reply

```
Enter msg text or null line to finish
```

You now type the lines of your message which is terminated when you enter a null line (i.e. just hit RETURN). You will then receive the READY message. If the recipient is logged in he will receive the message

```
YOU HAVE MAIL FROM userid
```

where userid is your username.

Subsequent messages to this user from yourself or from other users will be added to the end of his MESSBOX.

7.1.2 Mailing a File

Type

```
RMAIL userid fileid
```

where userid is the username of the person to receive the file, and fileid is the name of the file you wish to send. If not specified the filetype will default to MAIL and the filemode to A. You will receive the reply

```
      FILE FOR userid  RECEIVED AND STORED
```

If the recipient is logged in he will receive the message

```
      FILE FOR YOU RECEIVED FROM userid
```

at his terminal.

This message also goes into his message file.

7.1.3 Reading Your Message and Mail Files

To find out if there is any mail for you, simply type the command

```
      RMAIL ( QUERY
```

In response to this command, MAILMAN tells you how many messages are in your MESSBOX. Most users will probably find it useful to have this command in their PROFILE EXEC. If there is nothing for you, you will be told by MAILMAN that YOU HAVE NO MAIL. To list the contents of your MESSBOX, type the command:

```
      RMAIL
```

To read a mail file at your terminal type the command

```
      RMAIL userid (READ
```

This will type the mail file from the specified user at your terminal.

The command

```
      RMAIL userid fn ft (READ
```

will copy the mail file from the specified user into a CMS file with the specified filename and filetype.

In neither case is the file erased after reading.

7.1.4 Erasing your Message and Mail Files

To erase a mail file from a specified user, type

```
      RMAIL userid (ERASE
```

To erase all your mail files type

```
RMAIL * (ERASE
```

To erase the contents of your MESSBOX type

```
RMAIL (ERASE
```

7.2 <u>The QUERY Command</u>

There is a CP QUERY command and a CMS QUERY command and in both cases the command can be shortened to Q. The command is processed by CMS or CP depending on the operand given.

7.2.1 The CMS QUERY Command

This is used to find out the characteristics of your virtual machine. Some of the commands you may find useful are:

Q SEARCH	this displays the search order of all your accessed disks.
Q DISK <mode>	this displays information about the disk represented by 'mode'; for instance, QUERY DISK A will tell you about your A-disk; its label, its virtual address, how it is accessed, its size, how full it is, how many files it contains, how many blocks are free. The command QUERY DISK, or QUERY DISK * displays the status of all your accessed disks.
Q FILEDEF	tells you what file definitions you have in effect. You are told the ddname, the device address and the filename and filetype of your file definitions. This can also be achieved by using the RL additional command QFI, which gives more information about each FILEDEF.
Q LIBRARY	this displays the names of all files with filetypes of MACLIB or TXTLIB that are to be searched. It is also possible to list these separately using QUERY MACLIB or QUERY TXTLIB.

There are many other functions of the CMS QUERY command which you may find useful. They are described in the CMS Command and Macro Reference.

7.2.2 The CP QUERY Command

This is used to display elements of your virtual machine configuration; it can also be used to tell you about your usage and about your SPOOLed files. Below are some of the more useful commands; the minimum abbreviation is shown in capitals.

Q Time	this displays, amongst other things, the connect and processor time used so far in the current terminal session.
Q TERMinal	this displays information about the options specified for your terminal. Amongst other things it will tell you what are your logical line editing symbols.
Q Files	tells you how many spool files there are in the system for your virtual reader, printer and punch.
Q Reader Q Printer Q PUnch	these commands display information about all your closed reader, printer and punch files respectively.
Q UR	this displays the status of all your virtual unit record devices, i.e. the reader, printer and punch.
Q STORage	tells you the size of your virtual machine.
Q LOGmsg	this displays the log messages that you receive automatically when you log in.
Q userid	tells you whether a particular user is logged in or not, and if he is, the device address of his terminal.
Q Names	this displays a list of all users currently logged in, with the device addresses of their terminals.

There are many other functions of the CP QUERY command available; they are all described in the CP Command Reference for General Users.

7.3 Hardcopy of Sessions

A useful facility is provided by the CP SPOOL command which enables you to obtain a hardcopy of part or all of a terminal session. Issue the command:

```
SPOOL CONSOLE START
```

and then subsequent commands will be recorded in a VM spool file. The log can be terminated by the command:

```
SPOOL CONSOLE STOP CLOSE
```

By default, the file will be directed to the VM system printer and printed overnight, but this routing can be changed e.g.

```
SPOOL CONSOLE START *
```

will start the log and the resulting file will appear in your own system reader when the log is stopped. It may then be read into a CMS file by the READCARD command.

7.4 Contacting User Interface Group

Members of User Interface Group may be contacted by using the RMAIL command to send messages to userid US (see section 7.1).

Comments on the CMS service in general may be passed to the relevant people by using the GRIPE command. This will mail your messages to the GRIPE machine via the RMAIL command. Type:

```
GRIPE

one or more messages terminated by a null line (return).
```

Chapter 8: WHERE TO GO FROM HERE

8.1 EXEC Files

An EXEC file is a CMS file which has filetype EXEC and contains executable statements. These statements may be CP or CMS commands or special EXEC language control statements. The EXEC language allows the user to control the order of processing of statements within the EXEC file, to output messages to the terminal, to read the values of variables from the terminal, to examine the number of arguments passed to the EXEC, to check the return code from a CP or CMS command, and many other functions.

An EXEC file is invoked either by using the EXEC command or by typing its filename. This latter method gives the user the ability to override system commands. A useful EXEC file is the PROFILE EXEC which is normally invoked automatically when a user logs on.

A special type of EXEC file called an edit macro may be used within the editing environment. This gives users the ability to write their own edit commands. Details of several edit macros available to all users are given in section D3 of the RL VM User's Reference Manual.

For more detailed information about EXEC files see the CMS User's Guide, Section 6 and Part 3.

8.2 Temporary Disks

If a user's minidisk is nearly full, temporary space can be obtained by defining a temporary minidisk, usually referred to as a TDISK. Once defined and formatted a TDISK remains available for the rest of the user's logged-on session. However, on logging out, everything on the TDISK is lost. Files which need to be kept should be copied on to a permanent minidisk before logging out. For details of how to define and use a TDISK, see the TDISK command in the RL VM User's Reference Manual or Section 1 in the CMS User's Guide.

8.3 Update Files

Update files provide a means of making changes to a CMS file without affecting the text of the original file. An update file is an independent file with filetype UPDATE which contains control statements specifying the updates to be made to the source file. Any update file can be applied to any source file. This provides a mechanism by which several users can make modifications to the same source file, and several update files can be used at the same time. When a file is modified by an update file an update log is produced showing the changes that have been made.

There are two serious drawbacks to the use of update files:

(a) As there is no link between the update file and the source file, any changes made to the source file are likely to invalidate any update files. Consequently, if several users have update files which are applied to the same source file it would be almost impossible ever to change that source file and still keep the updates in step.

(b) Update files are not produced automatically by the editor but have to be created as separate files. The update instructions are very limited and have a completely different syntax from the edit commands.

For further details of how to create and use update files see the CMS User's Guide, Section 13, pp.252-end.

APPENDIX A: REFERENCES

This appendix contains a list of the manuals referenced in the body of this manual.

1 CMS User's Guide

2 CMS Command and Macro Reference Manual

3 CP Command Reference for General Class Users

4 RL VM User's Reference Manual

5 Extended VM/370 Editor User's Reference Manual

6 The Waterloo Script Reference Guide

7 CIGAR