

Edinburgh
Regional
Computing
Centre

EMAS-3: User's Guide

A GUIDE TO THE USE OF THE
EDINBURGH MULTI–ACCESS
SYSTEM EMAS-3

Preliminary Edition
September, 1986

Editor's Foreword

I wish to record my thanks to Roderick McLeod for writing this Guide and to the members of ERCC who made contributions.

In order to be able to publish this Guide by October it was necessary to describe EMAS-3 as it was in April. Consequently this text is believed to be adequate even if incomplete. Over the next few months the text will be developed prior to publishing the next edition.

Neil Hamilton-Smith
September 1986

PREFACE

EMAS-3 is the name used for the implementation of the Edinburgh Multi-Access System which runs on computers with the IBM XA Architecture. This Guide tells you how to use EMAS-A and some of the programs which are available on the system. It also tells you where to obtain further information.

In writing the Guide I have received much assistance from members of the EMAS team, many of whom have contributed parts of the text. Apart from them I wish to record my thanks to Neil Hamilton-Smith without whose constant help and encouragement I would have found it hard to complete the work.

I would like to dedicate this Guide to the memories of two colleagues whose untimely deaths occurred during the early days of EMAS-3: Gordon Burns and Colin McCallum. Both of them made lasting contributions to the Edinburgh Multi-Access System.

Roderick McLeod
Tankerness, Orkney
July 1986

CONTENTS

Chapter			Page
	PREFACE		iii
1	INTRODUCTION TO EMAS-3		
	1.1	What is EMAS	1-1
	1.2	History of EMAS	1-1
	1.3	Structure of EMAS-3	1-1
	1.4	The EMAS-3 Subsystem	1-1
	1.5	Primary Characteristics of EMAS-3	1-2
		1.5.1 Multi-Access	1-2
		1.5.2 General Purpose	1-2
		1.5.3 Secure Filestore	1-2
		1.5.4 Virtual Machine	1-3
		1.5.5 Fluent Communications	1-3
		1.5.6 Ease of Use	1-3
2	GETTING STARTED WITH EMAS-3		
	2.1	INTRODUCTION	2-1
	2.2	OBTAINING ACCESS TO EMAS-3	2-1
		2.2.1 How to become an accredited User	2-1
		2.2.2 Username and Password	2-1
		2.2.3 The command PASSWORD	2-2
	2.3	USING AN INTERACTIVE TERMINAL	2-2
		2.3.1 PADs and TCPs	2-2
		2.3.2 Accessing EMAS-3 using a terminal connected to a PAD	2-3
		2.3.3 Logging-on	2-3
		2.3.4 Terminal Characteristics	2-3
		2.3.5 Simple rules for typing when using EMAS-3	2-3
		2.3.6 Prompts and type-ahead	2-4
	2.4	TYPING EMAS-3 COMMANDS	2-4
	2.5	COMMAND PARAMETERS	2-5
		2.5.1 Positional Parameters	2-5
		2.5.2 Keyword Parameters	2-6
		2.5.3 Prompted Parameters	2-6
		2.5.4 Obligatory and optional parameters	2-6
	2.6	LOGGING OFF	2-7
	2.7	INFORMATION ABOUT EMAS-3	2-7
		2.7.1 Printed Documentation	2-7
		2.7.2 On-line Documentation	2-7
		2.7.3 The command VIEW	2-7
		2.7.4 The command HELP	2-7

Chapter Page

2.7.5 The command ALERT 2-8
2.7.6 How to use VIEW 2-8
Getting back from a subsection 2-9
Extracting Material 2-9

2.8 The command SUGGESTION 2-9

2.9 Commands described in this Guide 2-10

3 GENERAL FILE UTILITY COMMANDS

3.1 INTRODUCTION 3-1

3.2 The EMAS-3 File System 3-1
3.2.1 Naming Files and Groups 3-2
3.2.2 Summary of file naming rules 3-3
3.2.3 Creating Groups 3-3
3.2.4 Using a Group as a Sub-Index 3-4
3.2.5 Partitioned files 3-5

3.3 The command FILES 3-5

3.4 CREATING, RENAMING AND DESTROYING FILES AND GROUPS 3-7
3.4.1 Creating Files 3-7
3.4.2 The command RENAME 3-8
Renaming a file 3-8
Renaming a group 3-8
3.4.3 The command NEWGEN 3-9
3.4.4 The command DESTROY 3-10
3.4.5 The command DESTROYGROUP 3-10
3.4.6 The command DESTROYSET 3-10

3.5 CONNECTING AND DISCONNECTING A FILE 3-10
3.5.1 The command DISCONNECT 3-11

3.6 TRANSFERRING OWNERSHIP OF A FILE 3-11
3.6.1 Offering a file to another user 3-12
3.6.2 Accepting a file from another user 3-12

3.7 SETTING ACCESS PERMISSIONS ON FILES AND FILE GROUPS 3-13
3.7.1 Setting access permissions on files 3-13
What permissions are needed? 3-13
'All files' permissions 3-13
'All users' permissions 3-13
Permissions for usergroups 3-14
Own permission and P(rotect) 3-14
Full permissions 3-14
Default permissions 3-14
Multiple permissions 3-15
Revoking permissions 3-15
Partitioned files 3-16
Summary of parameters for PERMIT 3-16
Interrogating permissions 3-16
3.7.2 Setting access permissions on file groups 3-16
3.7.3 How permissions are stored 3-17
3.7.4 Determining whether a user can access a file 3-17
Restricting access to a file 3-18

Chapter Page

3.8 COMMANDS RELATED TO BACKUP STORAGE 3-19
3.8.1 The commands CHERISH and HAZARD 3-19

3.9 COMMANDS RELATED TO ARCHIVE STORAGE 3-19
3.9.1 The command ARCHIVE 3-20
3.9.2 Obtaining a list of files in the archive store 3-20
3.9.3 Moving files from the archive store to the disc store 3-20
3.9.4 The command AWAITRESTORES 3-21
3.9.5 Destroying files in the Archive Store 3-21
3.9.6 Setting access permissions on archived files 3-22
'All files' permissions 3-22
'All users' permissions 3-22
Permissions for usergroups 3-23
Own permission 3-23
Default permissions 3-23
Multiple permissions 3-23
Revoking permissions 3-23
Restricting permissions 3-24
Summary of parameters for APERMIT 3-24

4 GENERAL FILE MANIPULATION COMMANDS

4.1 INTRODUCTION 4-1

4.2 File Structure 4-1

4.3 The command ANALYSE 4-1

4.4 The command COPY 4-3
4.4.1 Copying complete File Groups 4-3

4.5 The command CONCAT 4-3

4.6 The command LINK 4-4

4.7 The command CONVERT 4-4

5 EDITING CHARACTER FILES

5.1 INTRODUCTION 5-1

5.2 THE STANDARD SUBSYSTEM EMAS EDITOR - EDIT 5-1

5.3 The command EDIT 5-1

5.4 METHOD OF EDITING 5-2

5.5 THE EDITOR COMMANDS 5-2
Letter case within text strings 5-3
5.5.1 Commands to alter the position of the cursor 5-3
5.5.2 Command to insert text 5-4
5.5.3 Commands to delete text 5-5
5.5.4 Terminal output from the editor 5-6
5.5.5 The cancel command 5-6
5.5.6 Terminating an edit session 5-6
5.5.7 Preserving editing done so far 5-7

Chapter			Page
5.6	MORE ADVANCED FACILITIES IN THE EDITOR		5-7
	5.6.1	Command repetition	5-7
	5.6.2	The separator S	5-8
	5.6.3	Moving a section of text within a file	5-9
	5.6.4	Extracting part of a file	5-10
	5.6.5	Text layout commands	5-10
	5.6.6	The Spelling Checker	5-11
5.7	THE OPERATION OF THE EDITOR		5-11
		Command Summary	5-12
5.8	The command LOOK		5-12
5.9	The command RECALL		5-14
5.10	THE EDINBURGH COMPATIBLE CONTEXT EDITOR		5-14
5.11	CALLING THE EDITOR		5-14
5.12	USING ECCE		5-14
	5.12.1	The file pointer	5-15
	5.12.2	A simple subset of commands	5-15
	5.12.3	An example of the use of ECCE	5-16
	5.12.4	On-line HELP for ECCE	5-17
	5.12.5	Text location and manipulation commands	5-18
	5.12.6	A further example of the use of ECCE	5-19
	5.12.7	Command failure	5-20
	5.12.8	Character manipulation commands	5-21
	5.12.9	Breaking and joining lines	5-22
	5.12.10	The repetition command	5-22
	5.12.11	Monitoring commands	5-22
	5.12.12	Upper and lower case control commands	5-23
	5.12.13	Dealing with non-printable control characters	5-23
	5.12.14	Context specification - D, F, T and U revisited	5-24
5.13	PROGRAMMED COMMANDS		5-25
5.14	MACROS		5-27
	5.14.1	Extended macros	5-28
5.15	PRESERVING YOUR OPTIONS		5-28
5.16	RECOVERING FROM ERRORS		5-28
5.17	SECONDARY INPUT		5-29
5.18	SECONDARY OUTPUT		5-31
5.19	ANCILLARY FUNCTIONS		5-32
	5.19.1	Formatting your text	5-32
	5.19.2	The Spelling Checker	5-33
		The Q (Query spelling) command	5-33
		The O (Orthography) command	5-33
	5.19.3	The IMP syntax checker	5-33
	5.19.4	Calling EMAS commands from within ECCE	5-34
5.20	The command SHOW		5-34

Chapter Page

5.21 The command RECAP 5-34

5.22 SUMMARY OF ECCE COMMANDS 5-35
5.22.1 Simple commands 5-35
5.22.2 Special commands 5-36
5.22.3 Programmed command qualifiers 5-37

6 SOFTWARE PACKAGES on EMAS-3

6.1 Introduction 6-1

6.2 Terminology 6-1

6.3 Sources of programs, packages and libraries 6-1

6.4 Importing software yourself 6-1

6.5 Finding out about available Software 6-2

6.6 CONLIB - the Contributed Library 6-3

6.7 Importing Load Modules from IBM systems 6-3

6.8 Commercial use of software 6-3

6.9 Criteria to use when selecting Software 6-3

7 COMPILERS, OBJECT FILES AND PROGRAM LOADING

7.1 INTRODUCTION 7-1

7.2 COMPILING PROGRAMS 7-1
7.2.1 The commands IMP and FORTRAN 7-1
Examples of calling compilers 7-2
7.2.2 The command PARM 7-2
7.2.3 The command INITPARM 7-3

7.3 OBJECT FILES 7-3
7.3.1 The structure of an OBJECT file 7-3
7.3.2 Linking Object files together 7-4
7.3.3 Altering OBJECT files 7-5

7.4 EXECUTING THE CODE IN AN OBJECT FILE 7-5
7.4.1 The command RUN 7-5

7.5 PROGRAM LOADING 7-6
7.5.1 Loading a single object file 7-6
7.5.2 Locating a procedure 7-6
7.5.3 The directory search order 7-7
7.5.4 The command SEARCHDIR 7-7
7.5.5 The command LOOKFOR 7-8
7.5.6 The command REMOVEDIR 7-8
7.5.7 The command LOADPARM 7-8

Chapter Page

7.6 DIRECTORY FILES 7-9
7.6.1 Creating directory files 7-9
7.6.2 The active directory and the command ACTIVEDIR 7-9
7.6.3 The command INSERT 7-10
7.6.4 The command REMOVE 7-10
7.6.5 The command TIDYDIR 7-11

7.7 DYNAMIC LOADING 7-11
7.7.1 Running a program with dynamic references 7-12

8 IMP ON EMAS-3

8.1 INTRODUCTION 8-1

8.2 COMPILATION 8-1
8.2.1 The structure of an IMP source file 8-2
8.2.2 External data 8-2
8.2.3 Calling Fortran 8-2

8.3 EXECUTING IMP PROGRAMS 8-2
8.3.1 Unsatisfied References 8-2

8.4 LIBRARY ROUTINES 8-3
8.4.1 IMP System Library 8-3
8.4.2 Graphics and other libraries 8-3

8.5 ACCESSING EMAS COMMANDS FROM IMP PROGRAMS 8-3

8.6 ACCESSING OTHER EMAS FACILITIES FROM IMP 8-3
8.6.1 Use of name type parameters 8-4
8.6.2 Setting the prompt text 8-4
8.6.3 Detecting multi-character Interrupts from the Interactive Terminal 8-5
8.6.4 Obtaining information about the Subsystem 8-5
8.6.5 Checking for the existence of a file 8-6

8.7 IMP INPUT/OUTPUT 8-6
8.7.1 Linking channel numbers to files and devices 8-7
8.7.2 Character I/O 8-7
Default stream definitions 8-8
8.7.3 Sequential Binary Files (SQFILES) 8-8
8.7.4 Direct Access Files (DAFILES) 8-8
8.7.5 Store Map Files 8-9

8.8 EFFICIENCY OF IMP WHEN USED WITH EMAS 8-9

8.9 IMP PROGRAMMING AIDS 8-10
8.9.1 The SOAP Formatting Program 8-10
8.9.2 IMP80 Syntax Checker 8-10

9 FORTRAN ON EMAS-3

9.1 INTRODUCTION 9-1

9.2 COMPILATION 9-1

Chapter Page

9.3 PROGRAM LOADING AND EXECUTION 9-2
9.3.1 Subroutine linking 9-2
9.3.2 Data linking 9-2
9.3.3 Running Fortran programs 9-3
9.3.4 System library 9-3
9.3.5 Mathematical, Graphics and other libraries 9-3

9.4 ACCESSING EMAS-3 FACILITIES 9-3
9.4.1 Use of CHARACTER variables 9-3
9.4.2 Language Independent Interface 9-4
9.4.3 Setting the prompt text 9-4
9.4.4 Detecting Interrupts from the Interactive Terminal 9-4
9.4.5 Calling EMAS commands 9-5
9.4.6 Checking the existence of a file 9-5
Obtaining information about the Subsystem Environment 9-5
9.4.7 The subroutine EMAS3UINFI 9-6
9.4.8 The subroutine EMAS3UINFS 9-6
9.4.9 Calling IMP 9-6

9.5 INPUT/OUTPUT 9-6
9.5.1 File types 9-6
9.5.2 Use of DEFINE with Fortran 9-7
9.5.3 Sequential input 9-7
9.5.4 Sequential output 9-8
9.5.5 Direct access files 9-8
9.5.6 Default definitions 9-9
9.5.7 Effect of OPEN on definitions 9-9

10 PASCAL ON EMAS-3

11 THE C PROGRAMMING LANGUAGE ON EMAS-3

12 OTHER PROGRAMMING LANGUAGES ON EMAS-3

12.1 INTRODUCTION 12-1

12.2 DESIGN AIMS FOR PROGRAMMING LANGUAGES ON EMAS-3 12-1

12.3 PROGRAM PORTABILITY 12-1

13 PROGRAM ACCESS TO DATA

13.1 INTRODUCTION 13-1

13.2 Files used for holding data 13-1

13.3 CHARACTER files 13-1

13.4 DATA files 13-2
13.4.1 The structure of FIXED format DATA files 13-2
13.4.2 The structure of VARIABLE format DATA files 13-3
13.4.3 The format of UNSTRUCTURED DATA Files 13-3

13.5 Linking programs to files and devices 13-3

Chapter Page

13.6 The DEFINE Command 13-3
13.6.1 The CHANNEL parameter 13-4
13.6.2 The FILE or DEVICE parameter 13-4
13.6.3 The SIZE parameter 13-5
13.6.4 The FORMAT parameter 13-5
13.6.5 Listing current definitions 13-5

13.7 Accessing data on magnetic tape 13-6

13.8 Calling DEFINE from within a program 13-6

13.9 The command CLEAR 13-6

13.10 Choosing a unique channel number 13-6

14 INTERACTIVE TERMINALS

14.1 Introduction 14-1

14.2 The connection between your terminal and EMAS 14-1
14.2.1 PADs 14-1
14.2.2 Terminals connected via modems 14-1
14.2.3 Terminals connected to other networks 14-2

14.3 Controlling the PAD 14-2
14.3.1 Direct control of the PAD 14-2
14.3.2 Control of the PAD using the SETMODE command 14-2

14.4 Telling EMAS about your Terminal 14-3

14.5 OPTION settings related to Interactive Terminals 14-3

14.6 Interactive Terminal Interrupts 14-4
14.6.1 Standard method of generating interrupts 14-4
14.6.2 Shorthand method of generating some single character interrupts 14-4
14.6.3 Meanings of single character interrupts 14-5
14.6.4 Multi-character interrupts 14-5

14.7 Reviewing your terminal conversation 14-5

14.8 Messages from other processes 14-6

14.9 Suppressing Messages 14-6

14.10 The TELL command 14-6

15 FILE INPUT AND OUTPUT

15.1 INTRODUCTION 15-1

15.2 Card and Paper Tape Input 15-1

15.3 Input from a Microcomputer 15-1
15.3.1 XTALK 15-1
15.3.2 KERMIT 15-2

Chapter Page

15.4 Inputting data from magnetic tape 15-2

15.5 Inputting data from other Computer Systems 15-2

15.6 Terminology - Documents and Queues 15-3
15.6.1 Finding out about your Documents 15-3
15.6.2 Output Device Names 15-3

15.7 Outputting your files 15-3
15.7.1 The command LIST 15-4
15.7.2 The command SEND 15-5
15.7.3 The command DELIVER 15-6

15.8 Interrogating the Device Queue 15-6
15.8.1 The command OUTPUT 15-6

15.9 Deleting Queued Documents 15-6
15.9.1 The command DELETEDOC 15-6

15.10 Transferring Files to and from other Computer Systems 15-7
15.10.1 Information about File Transfers 15-7
15.10.2 The command HOSTS 15-7

15.11 Remote devices 15-7
15.11.1 The command NLIST 15-7
15.11.2 The command NDOCUMENTS 15-8
15.11.3 The command NDELETEDOC 15-8

15.12 The command MAIL 15-8

16 RUNNING WORK IN BACKGROUND MODE

16.1 INTRODUCTION 16-1

16.2 Creating a Job File 16-1

16.3 The command DETACH 16-1

16.4 Checking the progress of your Batch Jobs 16-3

16.5 Deleting a job in the Batch Queue 16-4

16.6 Setting up initial conditions in a Batch job 16-4

17 ADVANCED USE OF COMMAND LANGUAGE

17.1 INTRODUCTION 17-1

17.2 Calling Commands 17-1

17.3 The Command History Mechanism 17-3
17.3.1 Reissuing earlier commands 17-3
17.3.2 Displaying stored command lines 17-3
17.3.3 Repeating commands with corrections 17-4
17.3.4 Other features of the command history mechanism 17-4

Chapter Page

17.4 The OBEY command 17-5

17.5 The Command Macro Scheme 17-5

17.6 Abbreviating Command Names 17-6

17.7 The command CPULIMIT 17-6

17.8 The command METER 17-6

17.9 The command OPTION 17-7

18 HOW TO WRITE YOUR OWN COMMANDS

18.1 Introduction 18-1

18.2 Command structure 18-1

18.3 The Parameter Acquisition Mechanism - PAM 18-2

18.4 Typical PAM calls 18-3

18.5 The PAM vector 18-4
18.5.1 Prompt field 18-4
18.5.2 Qualifier field 18-4
18.5.3 Default field 18-4
18.5.4 Help field 18-5
18.5.5 Keyword field 18-6
18.5.6 Missing PAM vector fields 18-6

18.6 Getting a string parameter 18-6
18.6.1 Tags for string qualifiers 18-7

18.7 Getting an integer parameter 18-8
18.7.1 Tags for the route qualifier 18-9

18.8 Getting a long real parameter 18-10

18.9 Setting a Return Code 18-10

18.10 Additional PAM routines 18-11

18.11 Commands written in other languages 18-11

18.12 User-provided help routines 18-12

18.13 User-provided checking routines 18-12

19 STORE MAPPING

19.1 INTRODUCTION 19-1

19.2 PRINCIPLE OF OPERATION 19-1

19.3 FILES FOR DIRECT MAPPING 19-1
19.3.1 Creating a store map file - the command NEWSMFILE 19-1

Chapter Page

19.4 USING STORE MAPPING IN IMP 19-2
19.4.1 Linking the file to a data structure within an IMP program 19-2
19.4.2 Effect of accessing the array 19-3
19.4.3 Mapping onto members of partitioned files 19-3
19.4.4 Closing mapped files 19-3
19.4.5 Examples of store mapping using IMP 19-4
19.4.6 Changing the size of a mapped file 19-6

19.5 USING STORE MAPPING IN FORTRAN 19-7
19.5.1 Outline of method 19-7
19.5.2 The command DATASPACE 19-7
19.5.3 Example of Fortran store mapping 19-7
19.5.4 Removing the link between a file and a COMMON block 19-8
19.5.5 Using more than one mapped COMMON block 19-8
19.5.6 Choice of access mode 19-9
19.5.7 Mapping onto members of partitioned files 19-9

19.6 STORE MAPPING AND PROGRAM PORTABILITY 19-9

19.7 CONCLUSION 19-9

20 PARTITIONED FILES

20.1 History of PARTITIONED Files 20-1

20.2 The structure of PARTITIONED files 20-1

20.3 Creating a PARTITIONED file 20-1

20.4 Naming individual members of a PARTITIONED file 20-2

20.5 Creating and altering members of PARTITIONED files 20-2

20.6 Reading from members of PARTITIONED files 20-3

20.7 Destroying and Renaming members 20-3

20.8 PARTITIONED files within PARTITIONED files 20-3

20.9 Relationship between PARTITIONED files and File Groups 20-4
20.9.1 The command GROUPTOPD 20-4
20.9.2 The command PDTOGROUP 20-5

21 MAGNETIC TAPE FILE HANDLING

21.1 INTRODUCTION 21-1

21.2 Obtaining and Registering User Tapes 21-2

21.3 STANDARDS 21-2
21.3.1 Hardware Standards 21-2
21.3.2 Identification 21-2
21.3.3 Software Standards 21-3

Chapter Page

21.4 HIGH LEVEL LANGUAGE ACCESS 21-3
21.4.1 Unlabelled Access - Processing Physical Files 21-4
21.4.2 Successive Processing of Tape Files 21-5
21.4.3 Character Codes 21-5
21.4.4 Error messages 21-5

21.5 UTILITY ACCESS 21-6
21.5.1 The command TAPEANAL 21-6
21.5.2 The command FROMTAPE 21-8
Summary of defaults 21-10
Simple examples 21-10
Derivation of output file name(s) 21-10
Multiple processing 21-11
More complex examples 21-11
21.5.3 The command TOTAPE 21-11
Multiple Processing 21-13
21.5.4 The command COPYTAPE 21-14

21.6 ERROR MESSAGES FROM RECORD LEVEL TAPE HANDLING 21-15

21.7 IMPORT/EXPORT OF DATA ON MAGNETIC TAPE 21-17

Appendix

1 THE STRUCTURE OF EMAS-3

A1.1 Introduction A1-1

A1.2 Virtual Machines A1-1

A1.3 Processes A1-1
A1.3.1 Supervisor processes A1-3
Local Controllers A1-3
Global Controller A1-3
A1.3.2 Paged processes A1-3
A1.3.3 Paged System processes A1-5

2 PAGING AND VIRTUAL MEMORY

A2.1 INTRODUCTION A2-1

A2.2 STORAGE ALLOCATION A2-1
A2.2.1 The conventional solution A2-1
A2.2.2 Further problems in a multi-access environment A2-2

A2.3 PAGING A2-2
A2.3.1 Page faulting and virtual memory A2-3
A2.3.2 The effect of paging on user programs A2-3
A2.3.3 Limiting page turns A2-4

3 CHARACTER CODES A3-1

Appendix Page

4 ERROR MESSAGES

4.1 Error messages at log-on A4-1

4.2 Subsystem Error Messages A4-1
4.2.1 General error messages relating to files A4-1
4.2.2 Other Subsystem failure messages A4-3
4.2.3 The source of failure messages A4-3

5 ENVIRONMENTAL QUERIES
Obtaining information about the Subsystem A5-1

6 GLOSSARY

BIBLIOGRAPHY

INDEX

CHAPTER 1
INTRODUCTION TO EMAS-3

1.1 What is EMAS

'EMAS' stands for Edinburgh Multi-Access System. It is the name of a large scale computing system which has, as its primary characteristic, the ability to provide computer power to many users simultaneously. Strictly, the name EMAS refers to the operating system - the software which, when run on suitable hardware, provides this multi-access computing environment. In practice the distinction between the operating system and programs which run on the system is somewhat blurred.

1.2 History of EMAS

In 1965 the University of Edinburgh was given the task of developing multi-access computing and during the next five years was one of the participants in the Edinburgh Multi-Access Project, which attempted to write a multi-access operating system for the English Electric Leo Marconi (later ICL) System 4-75 computer. Although the project failed to produce a usable system the experience gained enabled a small team from the University's Computer Science Department and the Edinburgh Regional Computing Centre to write the first effective version of EMAS. This eventually ran on a pair of System 4-75 computers and provided the main University computing service for many years.

In 1977 the EMAS operating system was re-implemented on the ICL 2900 range, using many of the ideas of the earlier version. The user appearance of the new system was virtually unchanged although it has developed considerably since then. EMAS 2900 has been used to provide services based in Edinburgh and Canterbury. These have provided large scale computing to four university communities (Edinburgh, Glasgow, Strathclyde and Kent) and to many research establishments.

In 1984 it was decided to move EMAS again, this time onto a type of computer which has the IBM XA architecture. Such computers are made by IBM and a number of other manufacturers. The first computer to be used for an EMAS-3 service is an Amdahl V7. On the occasion of this move it was decided to make significant changes to the user appearance of the system. These resulted from suggestions from users, particularly those who had experience of using other multi-access systems.

1.3 Structure of EMAS-3

It is not necessary to understand the structure of EMAS in order to make effective use of it. If you are interested in finding out what goes on 'behind the scenes' you should read Appendix 1.

1.4 The EMAS-3 Subsystem

The word 'Subsystem' has, for many years, been used as a name for that part of EMAS which comprises the user interface software. This is the software which you use when you or your program request services from EMAS. Whilst 'Subsystem' is not an ideal title, it has become an accepted part of the vocabulary of EMAS users.

For a time the title was expanded to 'The Edinburgh Subsystem' - to indicate that this was but one of several subsystems. Although the structure of EMAS lends itself to having different subsystems for different groups of users, in reality the vast majority use the standard subsystem, the Subsystem.

This Guide is, primarily, a manual about the EMAS-3 Subsystem. If you are interested to know how the Subsystem relates to the rest of EMAS you should read Appendix 1.

1.5 Primary Characteristics of EMAS-3

1.5.1 Multi-Access

EMAS is designed to provide computer power to many people at one time. The number depends upon the power of the computer on which the system is run and the nature of the work the users wish to do.

1.5.2 General Purpose

The system is designed to be able to cope with a wide range of tasks. It can handle programs such as text editors which make small demands on the computer and large packages which make heavy demands on the processor memory and filestore.

In particular it is designed to accommodate the varied load imposed by each user. For example if you run an interactive statistics package you will, at times, carry out large computations. A few seconds later you may be making no demand on the system at all, whilst you consider your next response in the light of the results of the computation. A minute later you may decide to scan through a file of many thousands of records to identify cases which fit in with some criteria.

The aim of the system is to accommodate your varying needs as quickly as possible without tying up resources, such as memory, unnecessarily.

1.5.3 Secure Filestore

EMAS provides a secure, long-term filestore. Its facilities for looking after your files are described in Chapter 3. In particular EMAS:

* protects your files from malicious or unintentional interference by other users.

* allows you to share any, or all, of your files with other users or groups of users, if you so wish.

* protects your files from corruption in the event of a hardware or software fault.

* provides an Archive store for files which you are not using at the moment.

* frees you from having to concern yourself with details of the filestore. For example you do not have to know which magnetic tape contains the latest backup of your precious files. The system attends to this sort of detail.

1.5.4 Virtual Machine

EMAS distributes its resources by providing a 'virtual machine' for each user. This includes an enormous virtual memory which is larger than the actual memory contained in the computer on which the system runs. This means it must be far larger than the share of the memory which could be available if the memory was shared between the active users. This fictional memory is provided by the paging hardware and software and is described in Appendix 2.

1.5.5 Fluent Communications

EMAS is connected to a communications network which provides gateways to other networks in the UK and world wide. This means that you can access EMAS from many locations and you can transfer files, and electronic mail, to and from a large number of institutions. Although EMAS itself is not used widely outwith Edinburgh, its communications interfaces conform to International standards.

1.5.6 Ease of Use

Most computer systems claim to be 'User Friendly' and EMAS is no exception. Its subsystem is designed to suit new users by providing on-line Help information at appropriate places. It is also convenient for experienced users. As an example the commands on EMAS are mainly English words, rather than the cryptic abbreviations used on some systems. On the other hand, experienced users can use the ALIAS mechanism to provide abbreviations for command names which are meaningful to them.

CHAPTER 2
GETTING STARTED WITH EMAS-3

2.1 INTRODUCTION

This chapter describes how to log-on to EMAS-3 and how to type commands to tell the system what you want it to do. You can find details of individual commands later in this Guide - there is a list of standard commands on the coloured pages at the end of this chapter. This chapter also tells you about other sources of information concerning EMAS-3.

2.2 OBTAINING ACCESS TO EMAS-3

Before you can start to use EMAS-3 you must become an accredited user of a computer service which runs EMAS-3. Further, you will need to arrange access to a suitable interactive terminal which is connected directly, or indirectly, to the Edinburgh University Communications Network - EDNET.

EMAS-3 is the name of an operating system. The name of the computer service, or HOST, on which it runs is EMAS-A. If it is run on further HOSTs you may be assigned to one of them. You will be told the name of the HOST when you are accredited. If this is other than EMAS-A you should use it in place of EMAS-A when logging on as described below.

2.2.1 How to become an accredited User

If you work in a department which already uses ERCC computing services then you should apply to your Head of Department or to the person appointed to liaise with ERCC. If your department or organization does not currently use ERCC services then you should contact the Service Support Unit at ERCC.

After the formalities have been completed you will be given a six character username and a password.

2.2.2 Username and Password

Your username serves as a unique code to identify you and your files. Normally it is made up of four letters which identify your department followed by two numeric digits to distinguish you from other members of your department. For example a user in the Physics department at Edinburgh University might have a username "EGNP88".

To make full use of EMAS you will need three passwords. The first one, the foreground password, is needed to enable you to log on to the system from an interactive terminal. The other two are used in connection with inputting files on cards or from other systems. Their use is described in Chapter 15. You will be given an initial foreground password when you are first accredited to use the system.

Note that usernames and passwords can be typed in upper case or lower case or a mixture of the two.

In common with other systems of this nature, EMAS is open to abuse by a group of

people known as "hackers". They obtain satisfaction by managing to break into computer systems, often by guessing passwords. In order to reduce the risk of other people using your username and, perhaps, altering your valuable files or running up charges for computer time you should take the following precautions:

* You should not reveal your password to anyone.

* You should change your password periodically by using the PASSWORD command (see below).

* You should choose an obscure password - your initials or the name of your department are too obvious and make the task of the hacker very easy.

For its part ERCC takes active steps to prevent this type of activity. However, the success of this depends on users following the advice given.

2.2.3 The command PASSWORD

Once you have logged on (see below) you can use the PASSWORD command at any time to change one or more of your passwords. Type the command PASSWORD and follow the instructions given. In order to change a password you will first have to type your current foreground password. This is to ensure that the command is being used by the owner of the username, rather than by a malicious person who has found an interactive terminal which has been left unattended whilst the owner has been logged-on.

2.3 USING AN INTERACTIVE TERMINAL

The primary method for accessing EMAS is via an interactive terminal. This may be one of the following:

* A Visual Display Unit (VDU), consisting of a keyboard and a screen.

* A hard copy terminal such as a Teletype, which produces its output on paper.

* A microcomputer running a suitable terminal emulation program.

Whichever type of terminal is used it must be connected to EDNET, the computer communications network which is operated by ERCC. If the terminal is situated in Edinburgh then it will probably be connected directly to EDNET. Otherwise it can access EDNET, and hence EMAS, via another network. User Note 69 explains how to access EDNET from locations outside ERCC.

If you do not have access to a suitable terminal in your department you can use one of the public terminals provided by ERCC in many locations in the University.

2.3.1 PADs and TCPs

The interactive terminal will be connected to EDNET by either a PAD or a TCP. More information about PADs is given in Chapter 14. TCPs are being replaced by PADs throughout the network, so this Guide does not contain information about accessing EMAS via a TCP. If you are still using a TCP you will find more information about using them in User Note 52. In order to log-on to EMAS you need to know which you are using. If you are unsure you should check with the person responsible for the terminal.

2.3.2 Accessing EMAS-3 using a terminal connected to a PAD

After checking that the terminal is switched on and is 'online' you should press the <RETURN> key a few times. You should then see the name of the PAD displayed followed by the prompt:

```
PAD>
```

You should reply:

```
CALL EMAS-A
```

and then press the <RETURN> key.

You should now see a message saying you are connected to EMAS-A, and then be asked for your username and password. If this does not happen or if you get a network failure message which you do not understand you should seek help from the person responsible for the terminal, or the ERCC Advisory service.

2.3.3 Logging-on

You should now type your username and password in response to the prompts. Terminate each by pressing the <RETURN> key. You will notice that your password is not echoed on the screen. This reduces the risk of its being discovered by someone else. You should now see an introductory message which confirms the identity of the Host computer system and provides information about the status of the service. There may be a message advertising a meeting or a change in the service schedule, or telling you that you should read the ALERT information (see ALERT command below). Finally there should be the word "Command:" which is an invitation to you to type a command.

Log-on failure messages are explained in Appendix 4.

2.3.4 Terminal Characteristics

The precise behaviour of your terminal when you are using it to access EMAS depends upon three things:

* The terminal itself. There are many types of interactive terminal each having its distinctive features. For detailed information about the terminal you are using you should read the terminal manual.

* The type of concentrator it is connected to - a TCP or a PAD.
  Some information about PADs is contained in Chapter 14 of this Guide.
  Further information on TCPs can be found in User Note 52.

* EMAS-3 itself. Further information about the way in which EMAS communicates with interactive terminals is contained in Chapter 15.

2.3.5 Simple rules for typing when using EMAS-3

In spite of the differences mentioned above, certain operations are provided on all terminals:

* For some operations you have to press two keys - the Control (<CTRL>) key and a letter. You should do this by holding down the <CTRL> key and pressing the letter. This appears below as, for example, <CTRL+R>.

* Information is sent to EMAS in complete lines. When you have finished typing a line you should press the <RETURN> key.

* If you notice an error before you press <RETURN> you can correct it by pressing the <DEL> key repeatedly until the character concerned has been deleted. The result displayed by pressing <DEL> depends upon the type of terminal and concentrator you are using. In some cases it will delete the offending characters, leaving the cursor in position to re-type the text. In this case the final result will appear on the screen as if you had not made an error. Other terminals, especially hard copy terminals will display each character as you delete it - sometimes enclosing the deleted characters within back-slash (\) characters.

* If you have made corrections and wish to see the final state of the line before pressing the <RETURN> key, you should press <CTRL+R>. The corrected line will be displayed and you will be able to make further changes if required. Note that the prompt (see below) will not be displayed by this facility.

* If you decide to delete a whole line before pressing <RETURN>, you should press <CTRL+X>.

* If you wish to send 'End of File' to a program or command which reads its data from the interactive terminal you should type <CTRL+Y>.

2.3.6 Prompts and type-ahead

Whenever EMAS is awaiting information from you it will type a prompt at the start of a line. When you first log-on that prompt will be "Command:". It will change depending upon the nature of the command being used. For example when you are using the MAIL command the prompt will change to "Mail:".

When you have become experienced with using EMAS you will probably find it convenient to anticipate these prompts by typing ahead. Normally this will cause the prompt to be suppressed, but under some circumstances the prompt will appear even though you have already typed the related input. This slightly irritating feature can be avoided at the cost of waiting for the prompt rather than typing ahead.

2.4 TYPING EMAS-3 COMMANDS

The main way to tell EMAS what to do is to type commands in response to the prompt "Command:". The full details of the command language are contained in Chapter 17. Here are the most important rules:

* Commands can be typed in lower or upper case, or a mixture of the two.

* You must not include a space within the name of the command.

* If parameters follow the command on the same line, they must be separated from the name of the command by one or more spaces.

* After typing the command name, and any parameters, you should press the <RETURN> key.

Here are some examples of commands which can be typed without parameters:

Command: METER	This prints time, date and usage information. See also Section 17.8.

`Command: FILES`	This tells you the names of your files.
`Command: USERS`	This tells you how many people are using EMAS.

2.5 COMMAND PARAMETERS

There are three ways to provide parameters for an EMAS-3 command:

* On the same line as the command name itself, in the order defined by the command.

* On the same line as the command, in any order, each parameter being preceded by a keyword to indicate its purpose.

* In response to prompts provided by the command itself on subsequent lines.

When you are used to a command, the first method is usually the quickest and most convenient to use. The second method is particularly suited to commands which take a large number of optional parameters, of which you normally use only a few. The third method is intended for use when you do not know, or have forgotten the parameters needed for a particular command.

The three methods are described in the following sections.

2.5.1 Positional Parameters

Most commands only require one or two parameters and once you have learnt them you will probably find it most convenient to type the parameters on the same line as the command itself. Individual parameters are separated by commas. For example COPY is used to make a copy of a file. It takes two parameters - the name of the file to be copied and the name of the file into which it is to be copied. For example:

```
Command: COPY FILE1,NFILE
```

The command EDIT is used to alter the contents of a character file. It can take one or two parameters. The following are valid calls:

```
Command: EDIT PROG27

Command: EDIT MPV1,MPV2
```

When using positional parameters the commas are used to distinguish the separate parameters. If a parameter is to be omitted in a particular call you can show this by including the comma which would normally follow it. For example in the following call of the FORTRAN command the third parameter has been omitted:

```
Command: FORTRAN PROGS,PROGY,,PF
```

It is not necessary to use commas if the omitted parameter is the last one to be used. In this call both the third and fourth parameters are omitted:

```
Command: FORTRAN PROGS,PROGY
```

If a command requires more parameters than you can fit on a line then you can continue on the next line after a comma. For example:

```
Command: FORTRAN ERCC07:MYPROG,
PROGY,PLIST,PFAULTS
```

2.5.2 Keyword Parameters

If you cannot remember the order required for parameters then you can type the parameters in any order, preceding each with a keyword and an equals sign. This method can be combined with the positional method described above. It is particularly valuable for commands which allow for a large number of optional parameters, since it avoids the need to type commas to indicate that some have been omitted. For example the DETACH command takes 13 parameters. If you only want to use a few of these then it is convenient to use keywords. For example the eighth parameter controls the time at which the detached job will be run. You could type:

```
Command: DETACH JOB7,,,,,,,14.00.00
```

This can be written more readily as:

```
Command: DETACH JOB=JOB7,AFTER=14.00.00
```

Note that it is possible to combine the two methods, using positional parameters for the early ones and keywords for later ones:

```
Command: DETACH JOB7,AFTER=14.00.00
```

2.5.3 Prompted Parameters

If you omit (by choice or error) any obligatory parameters then the command will prompt for the missing values. You should reply with the value concerned. If you do not understand what a particular parameter is used for you should reply with a question mark. The command will then provide a brief explanation and invite you type the the parameter again.

A variation on this is to type the command followed by a question mark. In this case it will prompt you for all its parameters, both obligatory and optional. To get an explanation of the command itself you should type its name followed by two question marks. You will now get an explanation of the command itself followed by a prompt for each parameter in turn. Again you can reply with a question mark to any of these prompts to get an explanation.

If you decide to abandon a command when it is prompting for parameters you should press CTRL+Y.

2.5.4 Obligatory and optional parameters

Some parameters are obligatory, they must be provided each time you use the command. Others are optional. In the descriptions of commands throughout this Guide the distinction is indicated by underlining the obligatory parameters. When the command prompts for optional parameters (because you typed a question mark after the command name) it will give a default value in brackets after each prompt. For example command SEND uses a first parameter of "T#LIST" by default. If you type:

```
Command: SEND ?
```

Then the resultant prompt will be:

```
File(t#list):
```

If you wish to accept the default value then you only need press the <RETURN> key. If you want to use another value then you should type it and then press the <RETURN> key.

2.6 LOGGING OFF

When you have finished using EMAS you should log-off by typing the command STOP. This will free your interactive terminal for another user. If you fail to log-off but leave your terminal unused then the system will log you off after a period of about 30 minutes. QUIT is an alias for STOP.

2.7 INFORMATION ABOUT EMAS-3

Apart from this Guide you can obtain information about EMAS-3 from a variety of sources - printed documentation, on-line documentation and courses.

2.7.1 Printed Documentation

ERCC User Notes describe particular aspects of the various computer systems supported by ERCC. Some of these Notes are concerned with programs or packages available on several systems including EMAS-3. Others are just concerned with EMAS-3. When EMAS-3 first became available a series of User Notes was published to assist users who wished to move work from EMAS 2900. User Notes are available from the ERCC Advisory service. The complete list of current User Notes is published in User Note 0. This Note also contains an index to all the other Notes and this is a valuable pointer to information not covered in this Guide.

The *Computing Newsletter,* published monthly by ERCC, includes articles about new items of software which have been mounted on EMAS.

The Bibliography at the end of this Guide lists some other publications which may be of interest.

2.7.2 On-line Documentation

This is documentation which you can read on your interactive terminal, or optionally list on a line printer. It is accessed by the commands VIEW, HELP and ALERT. All three use the VIEW mechanism which is described below.

2.7.3 The command VIEW

This command can be used to find information about ERCC services including EMAS. See "How to use VIEW" below for details. To find a particular topic type the command

```
Command: VIEW
```

This will display a series of main topics from which one can be selected, giving sub-topics and so on until the required topic is found.

2.7.4 The command HELP

This command uses the VIEW mechanism to display information about EMAS facilities. It differs from VIEW in that the topic is specified as a parameter and the command attempts to locate the correct place in the VIEW information without the need to select sections and subsections. For example

```
Command: HELP FORTRAN
```

2.7.5 The command ALERT

```
ALERT
```

This command provides information about faults and changes in the EMAS service or system. It uses the VIEW mechanism to display the text. See "How to use VIEW" below.

2.7.6 How to use VIEW

The VIEW system is a mechanism for browsing through text. It is used in the HELP and ALERT commands, as well as being available through the command VIEW. This section describes some of the basic facilities - enough to allow you to use HELP and VIEW. More comprehensive documentation is available through VIEW itself. In the description which follows the word page refers to a complete screenful (about 20 lines) of text.

VIEW is designed to work on video or hard copy terminals, but browsing through text is, of course, quicker on a video. See TERMINALTYPE (Section 14.4) for how to tell the system what sort of terminal you are using - this makes sure that the text is presented tidily to you.

When any of the VIEWing commands starts up, it will present you with the first 20 lines or so of a file of text. There are many ways to use VIEW, but the essential commands are:

To get to the next page, press <RETURN>
To get to the previous page, type - (minus) and press <RETURN>
To exit from VIEW, type E (or Q) and press <RETURN>

Most VIEWable files are divided into sections. VIEW displays a page of text at a time. A page is part or all of one single section of the file. A section may have several pages.

The message `'...more'` displayed at the bottom of screen means that there is more in the present section!

You get to the next page by pressing the <RETURN> key.

(You always have to press the <RETURN> key to make VIEW do something.)

You can go directly to page n in a section by typing `/n` Thus `/1` takes you back to the start of the section.

/ by itself takes you to the last page of the section.

Using <RETURN> on its own for "next page", - for "previous page" and `/n` for "page n" you can leaf through a section or select any page at random (moving forwards or backwards at will).

If a file has many sections, then each section will have a number and also a name. You can go to the first page in any section by typing

section number followed by <RETURN>
or section name followed by <RETURN>

A section in a file may itself be divided into subsections. If you are VIEWing a complicated file, you may want to know where you are in it. The full number of the current page is displayed in the top right hand corner of the screen. A number like

means "page 3 in subsection 4 of section 12".

Subsection names are often more useful than their numbers. They are usually designed to be helpful, so that typing a meaningful word will present you with a section of that name whose text describes what the word stands for: so that, for instance, typing METER might produce a section whose name is METER and which describes the METER command.

Getting back from a subsection

If, for example, you are in section 2.1.1 and wish to get back to 2.1, you can of course type `2.1`

Alternatively, you can type `U` which will take you back up one level. You can get right back to the top of the file by typing `T`.

Extracting Material

You can extract material from the file you are VIEWing by typing

	F<filename>	e.g. F<FRED>
or	F<output device>	e.g. F<.LP23>

All the text of the current section and any subsections is output. If you specify the name of an existing file then its contents will only be overwritten if it is a CHARACTER file. Otherwise the F<> command will fail. You can append text to the end of an existing CHARACTER file by typing `F<filename-MOD>`.

VIEW offers further facilities, such as Alphabetic searching of suitably structured files, a facility to set "book-marks" and to return to them and a method of locating part of the text by keyword. For using the VIEW, HELP and ALERT commands you should not need to use keyword searching or alphabetic searching, so they are not described here.
You should reply "?" or "help" to the `View:` prompt to get a very condensed summary of commands to view. Part of the text accessible through VIEW is a much fuller explanation of VIEW, with instructions on preparing your own VIEWable files. If you just give the VIEW command you will easily find that documentation.
Full details can be found in User Note 9.

2.8 The command SUGGESTION

EMAS has been developed over many years, largely in response to suggestions from users. Many of these made informally as a result of personal contact between users and ERCC staff. Some come in letters or telephone calls. It is appropriate that EMAS should provide a simple method to submit suggestions using your interactive terminal. To use this mechanism simply type the command:

```
Command: SUGGESTION
```

and follow the instructions.

You should not use SUGGESTION for items requiring an urgent response. For these you should contact the ERCC Advisory service in the first instance. The

SUGGESTION command is mainly intended for your ideas on improvements and changes you would like to see made to EMAS.

2.9 Commands described in this Guide

Table 2.1 lists the EMAS-3 commands which are described in this Guide, divided into functional groups, and gives a brief summary of the purpose of each command. Table 2.2 lists the same commands in alphabetical order. Each table provides a pointer to the main description of the command in this Guide. Some commands are described more fully in separate documents. See the notes above about alternative sources of information about EMAS-3. You should also use the index of this Guide to find further references to the command.

Group	Command	Purpose	Section
General File Utilities	ACCEPT	Transfer file OFFERed by another user	3.6.2
	APERMIT	Allow access to archived file	3.9.6
	ARCHIVE	Mark file for transfer to archive store	3.9.1
	CHERISH	Mark file (or group) for backing up	3.8.1
	DESTROY	Destroy file (or group)	3.4.4
	DESTROYGROUP	Destroy group and contents	3.4.5
	DESTROYSET	Destroy a set of groups or files	3.4.6
	DISCARD	Destroy file in archive store	3.9.5
	DISCONNECT	Remove file from virtual memory	3.5.1
	FILES	Obtain complete or partial list of files in disc and archive stores	3.3
	HAZARD	Remove CHERISH marker for file or group	3.8.1
	NEWGEN	Alter contents of a shared file	3.4.3
	NEWGROUP	Create new file group	3.2.3
	OFFER	Mark file for transfer to another user	3.6.1
	PERMIT	Allow other users access to a file or group	3.7.1
	RENAME	Change the name of a file (or group)	3.4.2
	RESTORE	Copy a file from archive to disc store	3.9.3
	USEGROUP	Select default group	3.2.4
Type Specific File Utilities	ANALYSE	Obtain details of type, contents, access permission, etc. of a file	4.3
	CONCAT	Join two or more character files	4.5
	CONVERT	Convert a data file to a character file	4.7
	COPY	Copy a file (or a group)	4.4
	GROUPTOPD	Copy the contents of a group into a partitioned file	20.9.1

Group	Command	Purpose	Section
Type Specific File Utilities (continued)	LIST	List file on output device	15.7.1
	NEWPDFILE	Create new, empty, partitioned file	20.3
	NLIST	List file on device not directly accessible from EMAS-3 host	15.11.1
	PDTOGROUP	Copy the contents of a partitioned file into a group	20.9.1
	SEND	List file on output device and destroy it	15.7.2
Manipulating Data	CLEAR	Break link set up by DEFINE	13.9
	COPYTAPE	Copy files from one magnetic tape to another	21.5.4
	DATASPACE	Set up link between file and Fortran COMMON block	19.5.2
	DEFINE	Set up link between logical channel and particular file or output device, or get list of current links	13.6
	DEFINEMT	Set up link between logical channel and particular magnetic tape file	21.4
	FROMTAPE	Copy file from magnetic tape to disc store	21.5.2
	HOLDMT	Hold magnetic tape at last position	21.4.2
	NEWSMFILE	Create new file to be accessed via store mapping facilities	19.3.1
	RELEASEMT	Unload magnetic tape to free tape deck	21.4.2
	RESETLOADER	Reset loader tables to initial state	19.5.4
	TAPEANAL	Summarize contents of a magnetic tape	21.5.1
	TOTAPE	Copy file from disc store to magnetic tape	21.5.3
File Editing	ECCE	Edit character file	5.11
	EDIT	Edit character file	5.3
	LOOK	Examine contents of character file	5.8

Group	Command	Purpose	Section
File Editing (continued)	RECALL	Examine file containing record of interactive terminal I/O	5.9
	RECAP	Examine file containing record of interactive terminal I/O	5.21
	SHOW	Examine contents of character file	5.20
Compilers and associated commands	FORTRAN	Compile Fortran77 source file	7.2.1
	IMP	Compile IMP80 source file	7.2.1
	INITPARM	Specify initial settings for PARM	7.2.3
	LINK	Join two or more object files	7.3.2
	LOADPARM	Set options for loader	7.5.7
	MAKECOMMAND	Convert name of procedure to name of command	18.11
	MODIFY	Alter the contents of an object file	7.3.3
	PARM	Set compiler options	7.2.2
	PASCAL	Compile Pascal source file	10
	RUN	Execute program	7.4.1
Commands associated with Directories	ACTIVEDIR	Specify active directory	7.6.2
	ALIAS	Give alias name to a specified command, or remove all aliases associated with the command	17.6
	INSERT	Insert details of object file in directory	7.6.3
	LOOKFOR	Look for directory containing nominated command.	7.5.5
	NEWDIRECTORY	Create a new directory file	7.6.1
	REMOVE	Remove reference to object file from directory	7.6.4
	REMOVEDIR	Remove directory from search list	7.5.6
	SEARCHDIR	Add directory to search list	7.5.4
	TIDYDIR	Tidy directory file	7.6.5

Group	Command	Purpose	Section
Background Mode	AWAITRESTORES	Suspend job until files restored	3.9.4
	BATCH	Interrogate batch job queue	16.4
	DETACH	Put job into background job queue	16.3
Information and other commands	ALERT	Obtain information on recent changes in the service	2.7.5
	CPULIMIT	Set time limit for each command	17.7
	DELETEDOC	Remove document from queue	15.9.1
	DELIVER	Set text for heading of line printer output, etc., or get current text	15.7.3
	DOCUMENTS	Print information about documents in System queues	15.6.1
	HELP	Get advice on using Subsystem	2.7.4
	HOSTS	Find out about any host	15.10.2
	MAIL	Handle electronic mail	15.12
	MESSAGES	Inhibit or permit messages to the interactive terminal	14.9
	METER	Print usage information for current session	17.8
	NDELETEDOC	Delete document sent using NLIST	15.11.3
	NDOCUMENTS	Find out about document sent using NLIST	15.11.2
	OBEY	Execute a sequence of commands	17.4
	OPTION	Set Subsystem options, or get list of options in effect	17.9
	OUTPUT	Interrogate queue of output files	15.8.1
	PASSWORD	Change passwords	2.2.3
	QUIT	Terminate session	2.6
	SETMODE	Set characteristics of interactive terminal	14.3.2
	STOP	Terminate session	2.6
	SUGGESTION	Send suggestion to System Manager	2.8
	TELL	Send message to specified user, immediately or at his next log-on	14.10

Group	Command	Purpose	Section
Information and other information (continued)	TERMINALTYPE	Specify type of interactive terminal	14.4
	TRANSFER	Transfer file to or from another computer system	15.10
	TRANSFERS	Interrogate file transfer queue	15.10.1
	USERS	Print number of currently active users	2.4
	VIEW	Display pages of information on interactive terminal	2.7.3

Table 2.1 EMAS-3 Command Summary

Command	Section
ACCEPT	3.6.2
ACTIVEDIR	7.6.2
ALERT	2.7.5
ALIAS	17.6
ANALYSE	4.3
APERMIT	3.9.6
ARCHIVE	3.9.1
AWAITRESTORES	3.9.4
BATCH	16.4
CHERISH	3.8.1
CLEAR	13.9
CONCAT	4.5
CONVERT	4.7
COPY	4.4
COPYTAPE	21.5.4
CPULIMIT	17.7
DATASPACE	19.5.2
DEFINE	13.6
DEFINEMT	21.4
DELETEDOC	15.9.1
DELIVER	15.7.4
DESTROY	3.4.4
DESTROYGROUP	3.4.5
DESTROYSET	3.4.6
DETACH	16.3
DISCARD	3.9.5
DISCONNECT	3.5.1
DOCUMENTS	15.6.1
ECCE	5.11
EDIT	5.3
FILES	3.3
FORTRAN	7.2.1
FROMTAPE	21.5.2
GROUPTOPD	20.9.1
HAZARD	3.8.1
HELP	2.7.4
HOLDMT	21.4.2
HOSTS	15.10.2
IMP	7.2.1
INITPARM	7.2.3
INSERT	7.6.3
LINK	7.3.2
LIST	15.7.1
LOADPARM	7.5.7
LOOK	5.80.1
LOOKFOR	7.5.5
MAIL	15.12

Command	Section
MAKECOMMAND	18.11
MESSAGES	14.9
METER	17.8
MODIFY	7.3.3
NDELETEDOC	15.11.3
NDOCUMENTS	15.11.2
NEWDIRECTORY	7.6.1
NEWGEN	3.4.3
NEWGROUP	3.2.3
NEWPDFILE	20.3
NEWSMFILE	19.3.1
NLIST	15.11.1
OBEY	17.4
OFFER	3.6.1
OPTION	17.9
OUTPUT	15.8.1
PARM	7.2.2
PASCAL	10
PASSWORD	2.2.3
PDTOGROUP	20.9.2
PERMIT	3.7.1
QUIT	2.6
RECALL	5.9
RECAP	5.21
RELEASEMT	21.4.2
REMOVE	7.6.4
REMOVEDIR	7.5.6
RENAME	3.4.2
RESETLOADER	19.5.4
RESTORE	3.9.3
RUN	7.4.1
SEARCHDIR	7.5.4
SEND	15.7.2
SETMODE	14.3.2
SHOW	5.20
STOP	2.6
SUGGESTION	2.8
TAPEANAL	21.5.1
TELL	14.10
TERMINALTYPE	14.4
TIDYDIR	7.6.5
TOTAPE	21.5.3
TRANSFER	15.10
TRANSFERS	15.10.1
USEGROUP	3.2.4
USERS	2.4
VIEW	2.7.3

Table 2.2 EMAS-3 Commands (Alphabetical order)

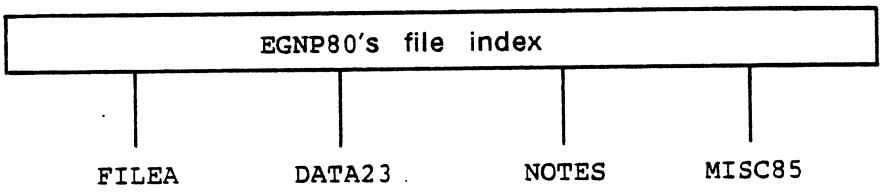
CHAPTER 3
GENERAL FILE UTILITY COMMANDS

3.1 INTRODUCTION

This chapter describes the EMAS-3 File system, and the rules for naming files. Then it describes the commands which operate on files as complete entities, without regard to their contents. In most cases they operate by making calls on the basic file handling services provided by Director (see Appendix 1). The different types of file provided by the Subsystem, and the commands specific to them, are described either in Chapter 4 or in the chapters following.

3.2 The EMAS-3 File System

The File System is a vital part of EMAS-3. It is managed by a part of the system called Director (see Appendix 1). Each accredited user has files in the on-line filestore, and may, additionally, have files in the Archive store. Information about your on-line files are contained in your own file index. If your username is EGNP80 the index can be represented thus:



These files would have names such as:

```
EGNP80:FILEA
EGNP80:DATA23   etc.
```

The full rules for naming files are given below.

On EMAS-3, in contrast to earlier versions of EMAS, it is also possible to *group* files together. A group is a named sub-index; its name is a common element in the names of files which are in the group.

For example, if you use EMAS for teaching and research you might choose to divide your files between two groups called TEACH and RESEARCH; and if your username is EGNP80 then the names of files used for teaching could be:

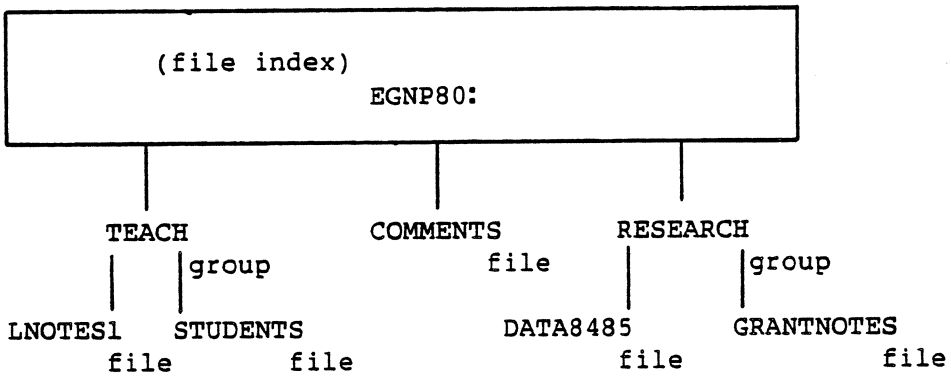
```
EGNP80:TEACH.LNOTES1
EGNP80:TEACH.STUDENTS
```

and your research files could include:

```
EGNP80:RESEARCH.DATA8485
EGNP80:RESEARCH.GRANTNOTES
```

You might also have a file of comments which is not in any group.

All this can be represented pictorially:



For obvious reasons, this type of filestore is sometimes described as 'hierarchical'.

Apart from enabling related files to be classified together, a group can be given *attributes,* such as access permission and cherish status, which then apply to all the files in the group.

3.2.1 Naming Files and Groups

Files which do not belong to a group are named thus:

username:localname

The localname consists of up to 11 alphanumeric characters of which the first must be a letter. Note the use of colon to separate the username from the localname, in contrast to the full stop used on earlier versions of EMAS. The following are valid EMAS-3 filenames:

```
ERCC98:FILEABC
CONLIB:GRAPHICSLIB
SUBSYS:SYSTEM
```

The groups TEACH and RESEARCH in the earlier examples belong to EGNP80 directly and not to other groups. The format of the name of a group is identical to that of a file, consisting of up to 11 alphanumeric characters (of which the first must be a letter). Within a particular group the files and groups can have any localnames, except that you cannot have a file and a group with the same name at the same level within a given branch of the hierarchy. But you can have files with the same names in different groups. For example you might choose to keep a NOTES file in each of your groups. Using the example above their full filenames would be:

```
EGNP80:TEACH.NOTES
EGNP80:RESEARCH.NOTES
```

It is also possible to have groups within groups. If, for example, you teach on two courses you could use a group for each course and it might be logical to make these groups belong to the group TEACH. They would have names such as:

```
EGNP80:TEACH.PH3
EGNP80:TEACH.PH4
```

and files associated with these courses would have names such as:

```
EGNP80:TEACH.PH3.STUDENTS
EGNP80:TEACH.PH3.EXAMDATES
EGNP80:TEACH.PH4.COURSENOTES
```

The structure of groups within groups can continue for many levels. For example,

```
EGNP80:ALPHA.BRAVO.CHARLIE.DELTA.ECHO
```

is a valid, if somewhat cumbersome, EMAS-3 filename. The only limit is that a full filename cannot exceed 255 characters.

3.2.2 Summary of file naming rules

- The full form of a filename is:

  username:[groupname.]localname

  The field in square brackets can appear zero or more times.
  (The square brackets themselves do not appear in the name.)

- The case of any letters within filenames is not significant.

- The username consists of six characters assigned by the System Manager.

- The username is separated from the rest of the filename by a colon.

- The groupname fields, if any, each consist of up to 11 alphanumeric characters, of which the first must be a letter.

- The groupnames are separated from each other and from the localname by full stops.

- The localname field consists of up to 11 alphanumeric characters, of which the first must be a letter.

See Section 3.2.4 below for the effect on filenames of using the USEGROUP command.

3.2.3 Creating Groups

Groups are created explicitly by using the command NEWGROUP, which takes one parameter, the name of the group to be created. For the examples above you would have to make calls:

```
Command: NEWGROUP TEACH
Command: NEWGROUP RESEARCH
Command: NEWGROUP TEACH.PH3
Command: NEWGROUP TEACH.PH4
```

An attempt to create a file or group in a non-existent group will produce a failure message. For example, if you type

```
Command: IMP TEST1, OBJ.TEST1
```

before creating a group OBJ then a failure will occur. Similarly, the last two NEWGROUP examples above would fail unless group TEACH had already been created.

A group is a file sub-index; it is not a file: creating a group does not create a separate file to contain a list of the group members, as happens on some other systems, for example UNIX. A group can exist even though it contains no files. Indeed, when it is first created it is empty.

It is useful to regard the file index as a 'super-group', which differs from other groups only in respect of its name: six alphanumeric characters, followed by a colon rather than a full stop.

A group has the following attributes:

- an owner
- zero, one or more superior group names
- a localname - i.e. a name to distinguish it from other files and groups at the same level
- a CHERISH status (see below)
- a set of access permissions, to its owner, to all other users and possibly to individual users or sets of users (see below)
- it may contain files
- it may contain groups.

3.2.4 Using a Group as a Sub-Index

Normally when a file is referenced by its localname alone, the assumption is made that it does not belong to any group. However, it is possible to nominate a group to take the place, temporarily, of one's file index, and file and group names specified thereafter are automatically prefixed by the nominated group's name. The command to nominate a group in this way is USEGROUP. For example:

```
Command: USEGROUP TEACH
```

Thereafter,

```
Command: COPY MARKSA, MARKSB
```

would be equivalent to

```
Command: COPY TEACH.MARKSA, TEACH.MARKSB
```

before TEACH was selected. Note that after TEACH was selected, this command would refer to files TEACH.TEACH.MARKSA and TEACH.TEACH.MARKSB!

The prefixing with the nominated groupname takes place for all filenames which do not contain ":" and of which the first two characters are not "T#". Thus to reference a file not in the currently selected group you must give its full filename, except that ":" is a permitted abbreviation for 'username:' for the current process owner. Thus:

```
Command: COPY MARKSA, EGNP80:MARKS
```

is the form required to copy a file out of the currently selected group, which may be abbreviated to:

```
Command: COPY MARKSA, :MARKS
```

The name of the group currently selected is displayed by the command

```
Command: USEGROUP =
```

The command

```
Command: USEGROUP
```

is used to cancel the current group selection, so that each localname reverts to being prefixed implicitly by 'username:'.

3.2.5 Partitioned files

The commands described in this chapter operate on complete partitioned files in just the same way as they do for any other files. For example you can PERMIT a partitioned file, and you can OFFER it. Chapter 20 explains how to use the commands which operate on individual members of partitioned files.

3.3 The command FILES

```
FILES pattern, options, out, type
```

This command is used to obtain a list of files and file groups belonging to you or another user. It takes up to four parameters.

`pattern` - The groups or files specified by the second parameter (see below), whose names match the pattern are included.

A simple pattern corresponds to a single group or file name, and is composed of three kinds of fields,

*	standing for 0 or more characters,
?	standing for exactly one character, and
other text	standing for itself,

and can have any number of such fields, e.g.

ABC	matches only ABC (one field),
AB*	matches AB,ABC,ABCD etc. (two fields),
AB?	matches ABC but not ABCD (two fields),
*AB*DE?	matches ABDEF and XABYDEZ but not ABDEXY (three fields).

A \ appearing at the start of a simple pattern negates the sense of the match, so that what matches is not included and what does not match, is.

Simple patterns can be joined, like group and file names, by the group separator character (a full stop), to give a pattern which matches over several levels of a hierarchical file name, thus

AB.*	matches all names of the form AB.name,
AB*.D*	matches ABC.D and ABC.DEF but not ABC.XYZ.DEF

Two group separators appearing together (or one appearing at the start of a pattern) means that 0 or more hierarchical levels may intervene, so that

AB*..D* (compare this with the previous example)
matches ABC.D (0 intervening)
and ABC.DEF (0 intervening)
and ABC.XYZ.DEF (1 intervening),

AB.. matches everything of the form AB.name.name..
to the bottom of the hierarchy, and

. matches everything to the bottom of the hierarchy.

Any such pattern can have one or more partitioned file member patterns tacked on the end of it, as

```
_simple pattern
```

Any partitioned files which have been included will be expanded and their member names matched against the simple pattern; this can be continued to several levels of partitioned file nesting by a pattern of the form

```
_simple pattern_simple pattern_simple pattern
```

or to an indefinite level of nesting by

```
__
```

which is analogous but less general than the .. construction described above, in that anything to the right of the first __ is ignored; thus

_A matches all files or groups at the current level and for those files which are partitioned files, matches only those member names which begin with A,

._ _ matches everything to the bottom of the hierarchy, and expands partitioned files to any depth of nesting.

A null pattern is treated as *, which matches any single name.

Before being applied, the pattern is prefixed with the current USEGROUP (see above); this means that if user EGNP88 has set a default group of PROJ17 then a pattern of *ABC is the same as a pattern of EGNP88:PROJ17.*ABC

A pattern can be prefixed by another user's number, e.g. ERCC04:AB* and if user ERCC04 has granted 'whole index' access permission, i.e. some permission to all of his files, the groups or files belonging to that user and which match the pattern will be included.

options - Specifies the files or groups to be included in the list. If omitted, all files and groups in the immediate (on-line) store are selected. Any combination of the following letters may be specified:

I files and groups in the disc file store (excluding T# files but including SS# files)
C CHERISHed files and groups in the disc file store
H HAZARDed (uncherished) files and groups in the disc file store
A ARCHIVEd files.

Six further codes are available which may be combined with the above:

S single spacing (print file names down the page); printing across the page is the default.

E print extra information (size and permissions); causes a one-line summary of each of the user's disc files to be printed, including T# files.

P print information about the user's file index as a whole, including access permissions for .ALL (see Section 3.7, below).

F files only

G groups only; if neither F nor G is used then both files and groups are included

R ARCHIVEd files on EMAS or BUSH which can be restored to EMAS-A by using the RECOVER command.

Any combination of the letters given above can be specified, in any order. Alternatively the parameter can be specified as "*", which is shorthand for immediate and archived files plus E and P.

On output a file or group name is preceded by "*" if it is CHERISHed and by "**" if it has been nominated for archiving. File group names are followed by a full stop to distinguish them from filenames. Files which are not CHERISHed and remain unused for a period (typically four weeks) are automatically destroyed. File names which are due to be destroyed shortly are preceded by '-'.

out - This may be either an output device code (.OUT by default) or an output file name. For example:

```
Command: FILES *S,,FILELIST
```

As a safeguard against overwriting a check is made that if the file already exists, it is a CHARACTER file.

type - The file type is only relevant to files, and if present will cause groups to be excluded. File types are selected by the following letters:

C character

O object

D data

N non-standard

P partitioned

Y directory

Any number of these may appear in any combination. If none appear all types are included.

3.4 CREATING, RENAMING AND DESTROYING FILES AND GROUPS

3.4.1 Creating files

Files can be created, renamed or destroyed only by their owners. Files are normally created as a by-product of certain commands being used. For example, the command EDIT can create a new character file. If this file is compiled by using the command IMP an object file may be created. If the output from ANALYSE is directed to a file, one will be created. COPY can be used to create a new file whose contents are identical to the data in some existing file. A few commands such as NEWPDFILE (Section 20.3), NEWSMFILE (Section 19.3.1) and NEWDIRECTORY (Section 7.6.1) have no other function than the creation of new files, but most files do not need to be explicitly created before they are used.

In general the following rule applies in relation to commands that create files implicitly:

> If a file already exists, and is of the same type as that being created then its contents will be overwritten. If it is of another type then the command will fail and the file will remain unchanged. If the file does not exist it will be created.

Note that you can provide additional protection for precious files by using the protected access permission (see Section 3.7, below).

3.4.2 The command RENAME

This command can be used to change the name of a file or group.

Renaming a file

```
RENAME file, newname
```

`file` must be the name of one of your files.

`newname` is the new name you want to give the file.

Note that both parameters will be affected by the currently selected group.

Examples:

```
Command: RENAME S1,S176
Command: RENAME ACCOUNTS.Y85,ACCOUNTS.OLDY85
```

RENAME can be used to move a file to a different place in the hierarchy. For example:

```
Command: RENAME ACC1986,ACCOUNTS.YEARLY.A1986
```

You cannot RENAME a file to have a name identical to another file or group (but you can have files in different groups with the same local name - `HOLD.S1` and `DATA.S1`).

You cannot RENAME a file whilst it is connected in another user's virtual memory, nor whilst it is on OFFER.

The CHERISH status and ACCESS permissions of a file are not affected by RENAME.

Renaming a group

```
RENAME group, newname
```

RENAME can also be used to rename a group, and hence all the members of the group, and members of them etc.

As with renaming files you can use RENAME to move a whole group to a different place in the hierarchy:

```
Command: RENAME :ACC1986,ACCOUNTS.Y1986
```

A group cannot be renamed to have an identical name to that of an existing file or

group. Further it cannot be renamed if any of its members (or members of members etc.) are connected in any virtual memory, or is on OFFER.

3.4.3 The command NEWGEN

NEWGEN ownfile, ownfile

This command is only of interest to users who provide files for other people. It is not possible to alter the contents of a file if it is connected in another user's virtual memory, in READ mode. This can happen when, for example, another user is running one of your programs. NEWGEN is used to circumvent this problem. The method is to create a file containing the new material with a different name from the current version. Then by typing

```
Command: NEWGEN newfile, existing file
```

the following actions occur:

* the contents of `newfile` replace the contents of the `existing file`
* the CHERISH status (see below) and access permissions of the `existing file` remain unchanged;
* any users currently accessing the `existing file` do not see any change - but continue to access the old contents until they disconnect the file (or log-off);
* the `newfile` disappears as a separate entity.

In the example below it is required to replace the version of ALERTMESS with a file which contains some additional text.

```
Command:EDIT ALERTMESS,TEMP
  Edit:I/The additional text
     /:/
  Edit:E

Command:NEWGEN TEMP,ALERTMESS
```

From this time on any user who connects ALERTMESS will find it contains the additional text, whereas any user who already has the file connected in their virtual memory would continue to use the unmodified version.

Notes

* The two parameters are complete filenames, and both are interpreted according to the USEGROUP setting.
* NEWGEN checks that the two files are of the same type.

3.4.4 The command DESTROY

This command can be used to destroy

* one or more files;
* one or more groups, so long as they contain no files or groups.

The command takes the names of the items to be destroyed as its parameter.

If more than one item is used they must be separated by ampersands:

```
Command: DESTROY ABC
Command: DESTROY TEMP & COBJ & BACL3PD_DFT
```

The names are interpreted according to the default group established by USEGROUP. If you type:

```
Command: USEGROUP GUIDE
```

and then type:

```
Command: DESTROY CHAP1 & CHAP18
```

then the files GUIDE.CHAP1 and GUIDE.CHAP18 would be destroyed.

The command will fail to destroy a specified file if it is connected in another user's virtual memory or if it is on OFFER (see below). Also, if a file is permitted to its owner with an access permission of RP (i.e. read only and protected) it cannot be destroyed. This can be used to protect a file from inadvertent destruction.

3.4.5 The command DESTROYGROUP

If you wish to destroy a group which still contains members then you can use the command DESTROYGROUP. This will destroy the group, any files or groups which it contains, down to the bottom level of each branch of the hierarchy. This command should be used with great care!

3.4.6 The command DESTROYSET

This command is used to destroy the set of groups or files which match a pattern (see the description of the 'pattern' parameter for the FILES command in Section 3.3).

All candidates for destruction are first displayed and you are asked whether you wish to destroy them all, none of them or be queried about each one individually. In the latter case each name is displayed and you are asked whether you wish to destroy it; you may respond `'yes','no','analyse'` or `'quit'`. In the case of `'analyse'` the same name is redisplayed after the analysis.

3.5 CONNECTING AND DISCONNECTING A FILE

Before any use can be made of the contents of a file, e.g. before a character file can be edited or an object file executed, it must be connected in the user's virtual memory. This operation is described in in Appendix 2. There is no general command for this purpose - connection occurs as a result of the use of a wide variety of commands or facilities. For example, a file is connected when:

* it is analysed by ANALYSE

* it is listed on the interactive terminal by using LIST

* it is edited;

* it is read by a FORTRAN program.

Normally, once a file has been connected, it remains connected for the rest of the

session - i.e. until the user logs off. There are, however, a number of commands which cause disconnection, and there is also an explicit DISCONNECT command. The following commands disconnect the file on which they are operating if it is connected at the time they are invoked:

DESTROY	RENAME
OFFER	PERMIT
SEND	

3.5.1 The command DISCONNECT

```
DISCONNECT filelist
```

This command can be used to disconnect one or more files from the user's virtual memory. It takes as its parameter the name of one or more files that are currently connected. The filenames are interpreted according to the currently selected group as specified by USEGROUP. The filenames should be separated by ampersands. The form

```
Command: DISCONNECT .ALL
```

can be used to disconnect all but essential process files.

There are several situations in which the command is useful:

* To protect a file. Since the file is no longer connected in the virtual memory it cannot be corrupted by programs being run by this user that might be faulty. In fact this form of corruption is unlikely, since user files are normally left connected in READ mode (and are therefore protected) when they are not currently being used for output.

  Note that the System automatically ensures that the disc copy of a connected file is kept up to date. There is thus no need to use DISCONNECT for this purpose. This does not mean, however, that the output file specified when a text editor is invoked (see Chapter 5) is kept up to date as the editing proceeds. For example, EDIT operates on a temporary copy, and the specified output file is not involved until the editing session is terminated or the file is preserved by use of the W command.

* To free the file for use by another user.

* To free space in the virtual memory. Although large the virtual memory can be filled during a session. To free some space the user should use DISCONNECT to disconnect some of the files that are no longer being used.

3.6 TRANSFERRING OWNERSHIP OF A FILE

The two commands OFFER and ACCEPT can be used to transfer a file from one user to another.

3.6.1 Offering a file to another user

```
OFFER ownfile, user
```

The owner of the file should use the command OFFER, which takes two parameters:

the name of the file to be transferred modified by the currently selected group, and the name of the user to whom it is to be transferred. For example:

```
Command: OFFER ABC,ERCC98
```

would offer the file ABC to user ERCC98.

Note that once a file is on offer it cannot be connected in any virtual memory, regardless of access permissions.

An OFFER can be revoked, if necessary, by using the command OFFER with only one parameter - the name of the file. For example:

```
Command: OFFER ABC
```

would revoke the offer of the file made to ERCC98 in the example above.

3.6.2 Accepting a file from another user

```
ACCEPT file, newname
```

The user to whom the file is offered can accept it at any time by using the command ACCEPT. This takes as its first parameter the full file name of the file to be accepted. For example, if the user OFFERing the file in the first example above had been WXYZ38 then the user ERCC98, to whom it had been OFFERed, would type

```
Command: ACCEPT WXYZ38:ABC
```

The effect of this would be to transfer the file ABC from user WXYZ38 to user ERCC98, giving it the new name ERCC98:ABC. This command will fail if user ERCC98 already has a file ABC. However, this problem can be overcome by typing a second, optional, parameter to ACCEPT, which is the new name to be given to the file. For example,

```
Command: ACCEPT WXYZ38:ABC,NEWABC
```

In this case the file will be transferred and given the new name ERCC98:NEWABC.

Note that both parameters are interpreted according to the currently selected group which has been specified by USEGROUP.

If the second parameter is omitted then the file will be given the same full name that it had originally, apart from its username. For example if user ERCC98 types:

```
Command: ACCEPT EGNP88:GRAHAM.PASCAL.TRIAL23
```

then the resulting file will be ERCC98:GRAHAM.PASCAL.TRIAL23. This transfer would fail unless ERCC98 already had a group called GRAHAM which contained a group called PASCAL. It would also fail if he already had a file (or group) called ERCC98:GRAHAM.PASCAL.TRIAL23.

3.7 SETTING ACCESS PERMISSIONS ON FILES AND FILE GROUPS

The command PERMIT can be used to set access permissions on individual files or on complete file groups. The command APERMIT (see Section 3.9.6 below) can be used to set access permissions on files in the archive store.

3.7.1 Setting access permissions on files

```
PERMIT filename, username, mode
```

The first parameter is the name of the file to be PERMITted (expanded as described in Section 3.2.4). The second is the name of the user who is to be allowed to use the file. The third indicates the modes of access which will be allowed to that user.

The mode parameter consists of one or more letters. Each letter indicates a certain mode of access. Only certain letters are permitted. The letters may occur in any order, but no letter may occur more than once. R stands for 'read'; W stands for 'write'. Other acceptable letters are described below. The default value for this parameter is R when permitting a file to another user. For example:

```
Command:PERMIT LUDU,ERCC14
```

would permit the file or group LUDU to ERCC14 in READ mode.

What permissions are needed?

Most uses of a file will need R access to it.

W access allows a user to overwrite the contents of another user's file, but not to change its size, nor to DESTROY it. This means that most programs, commands and utilities (including those supplied by the system) cannot change another user's file even if W access permission has been granted. In particular, it is not possible to EDIT the file. In fact, W access is most often useful for Direct Access and Store Map files.

'All files' permissions

It is possible to give a user access to all your files. To do this, you specify the filename as .ALL. Example:

```
Command: PERMIT .ALL,ZTBZ03,R
```

Note that access permissions given in this way apply to new files created *after* the PERMIT command is given, as well as to all files existing at the time of the command.

'All users' permissions

If you want, you can allow all users to have access, either to an individual named file or to all your files. Instead of a user name, you must specify .ALL, or ??????, or omit it altogether. Note that this does not affect your own access permissions to the file(s). What you are doing is properly described as setting 'everyone else's permission', which is abbreviated to EEP in the output from some commands and utilities. Examples:

```
Command: PERMIT HOLKAS,,RW
Command: PERMIT .ALL,ERCC99,R
Command: PERMIT TEMP,??????,R
```

Permissions for usergroups

The form ?????? meaning 'all users' is a special case of a 'usergroup name'.

You can specify a usergroup name to allow several users to access one (or all) of your files. A usergroup name is a user name in which some of the characters are replaced by question marks, thus: ER??0?. This usergroup would include ERCC08 and ERDB06, but not ECSC07 nor ERCC14. A usergroup includes any user whose name matches those characters in the usergroup name which are not question marks. Examples:

```
Command: PERMIT PUPTON,ER??0?,RW
Command: PERMIT BAWLEY,GRY???,R
```

Own permission and P(rotect)

You can also specify your own user name in a call of PERMIT (but only for a named file, not for 'all files'). This has special effects which are *not* achieved by specifying 'all users', nor even a usergroup which includes your own name. When one of your files is first created, you have full access permissions to it, so you would normally use PERMIT to restrict your access to it - typically, to prevent accidentally DESTROYing or overwriting the file. To prevent overwriting, you might do

```
Command: PERMIT VITAL,myname,R
```

or, to protect against DESTROYing as well,

```
Command: PERMIT AEONS,myname,PR
```

This demonstrates the use of P (meaning 'protect') to inhibit DESTROY. You can only specify P when you are giving yourself permission to a file - it is not valid as a permission to other users (who cannot DESTROY your file anyway). P on its own is taken to mean PR.

Note that P will prevent you from DESTROYing a file, but you will still be able to EDIT it if you have allowed yourself W permission. COPY also needs W access to overwrite a 'protected' file, but NEWGEN can replace it with another file regardless of its other permissions. Some other commands, utilities and programs may refuse to handle a 'protected' file when they would otherwise destroy it or replace it with a new version. However, you can still lose the file by using the commands SEND or OFFER.

Full permissions

The fullest possible access permissions are RW for an individual file, and R for 'all files'. Examples:

```
Command: PERMIT FILE1,GRAC??,RW
Command: PERMIT ARTIMON,LRHB21,RW
```

Default permissions

If you leave out the last parameter in a PERMIT command, then R permission is assumed, unless the user name is your own name, when RW (i.e., full permissions) is assumed. Thus, if you want to permit a file to someone else, then calling `PERMIT filename,username` is nearly always sufficient. `PERMIT filename,myname` is a convenient way to restore full access to one of your own files after you have restricted your own access - for example, when you actually want to DESTROY it. Examples:

```
Command: PERMIT KRAEK,GRVA02
Command: PERMIT ,ECSC??
```

If you give the command PERMIT with no parameters at all, it will have no effect (and it will not give you any error message). This avoids the rather dramatic, and probably unintended, effect of giving all users R access permission to all your files. If you really want to do that, you can do, for instance,

```
Command: PERMIT .ALL
```

Multiple permissions

If you do

```
Command: PERMIT BONE,PTFE25,W
Command: PERMIT BONE,PTFE25,R
```

then the file BONE is accessible to user PTFE25 with permission R, not RW. A PERMIT command supersedes all previous permissions granted for the same file and user (or usergroup). The same thing is true for "all files" permissions.

However, that does not prevent a file from accumulating many different permissions for different users and usergroups. Examples:

```
Command: PERMIT .ALL,E?????,R
Command: PERMIT TOKEN,GASP24,W
Command: PERMIT TOKEN,GASP25,W
```

In this case, EJRM24 could read TOKEN (because the first PERMIT allows him to read any file), and GASP24 and GASP25 can both write into it.

Revoking permissions

Sometimes one needs to withdraw a permission that has already been granted. This is done by specifying the single letter C (for 'cancel') as the third parameter to PERMIT. The first two parameters must, of course, be the same as in the command which established the permission (although the various synonyms of .ALL are not distinguished). Thus

```
Command: PERMIT TGANSL,RAMC42,R
```

is not cancelled by

```
Command: PERMIT .ALL,RAMC42,C
```

nor by

```
Command: PERMIT TGANSL,RAMC??,C
```

although .ALL (all files) includes TGANSL and RAMC?? includes RAMC42. You would have to give

```
Command: PERMIT TGANSL,RAMC42,C
```

but

```
Command: PERMIT COGG,,R
```

would be cancelled by

```
Command: PERMIT COGG,??????,C
```

Incidentally, if you DESTROY a file, there is no need to cancel the permissions for it. They are lost with the file, and will have to be set up afresh if you create a new file with the same name. You should also note that you cannot cancel your permissions to one of your own files.

Partitioned files

You can set access permissions for the whole of a partitioned file, but not for its members individually. The members have the same permissions as the whole partitioned file.

Summary of parameters for PERMIT

	Set	Revoke	Debar
Specific file			
self	file,myname,perms#1	-	-
user	file,user,perms#2	file,user,C	file,user,N
usergroup	file,group,perms#2	file,group,C	file,group,N
all users	file,,perms#2	file,,C	-
All files			
self	-	-	-
user	,user,perms#3	,user,C	,user,N
usergroup	,group,perms#3	,group,C	,group,N
all users	,,perms#3	,,C	-
	Acceptable:	Default:	
perms#1	P, W, R	RW	
perms#2	W, R	R	
perms#3	R	R	

Interrogating permissions

It is apparent by now that the matter of permissions is not simple. There are, as we have seen, several classes of permissions: for specific files and for all files; for the owner, for other named users, for usergroups and for all users. The best way to find out how they work is to use the PERMIT command on one of your own files, and then use the command `ANALYSE filename,P` to see a list of the permissions that you have set up.

3.7.2 Setting access permissions on file groups

It is also possible to use PERMIT in a similar way for a whole file group.

Notes

* You can set a permission for individual users, for usergroups and for everyone else. You cannot set a permission on a file group for yourself.

* The permissions set for a file group can affect all files in the group, all groups in the group and hence, all files in them. However, when determining the permission in a particular instance, Director examines the permissions on the file itself first. Then on its parent group, and so on up to the highest group. This is explained more fully in Section 3.7.4.

3.7.3 How permissions are stored

You will observe that the various permissions are not all muddled up together. The system keeps the different kinds of permissions in separate places -

Permissions for individual files:	
Owner's permission	OWNP ('own permission')
Other named users	in a list
Usergroups	in the same list
All other users	EEP ('everyone else's permission')
Permissions for file groups:	
Other named users	in a list
Usergroups	in the same list
All other users	EEP ('everyone else's permission')
Permissions for all files:	
Owner's permission	(no such thing!)
Other named users	in a list (not the same as the list for individual files)
Usergroups	in the same 'all files permissions' list
All other users	EEAFP ('everyone else's all files permission')

3.7.4 Determining whether a user can access a file

The question arises of how the system decides what access permissions are allowed to a particular user. The rule goes like this:

1. If the user is the owner of the file, then OWNP applies and no other permissions are relevant (which explains why there are no 'own permissions' to 'all files').

2. If the user is not the owner, then the individual file's permission list is scanned for the user's name (usergroups are ignored at this stage). If his name is found, then the associated permission applies and no other permissions are relevant - even if the permission found is inadequate for what he wants to do. Note that this stops the search, so that a more generous permission in, for instance, the 'all files' permission list, will *not* be found.

3. If the user's name was not found at stage 2, the individual file's permission list is scanned again for a usergroup name which includes the user's name. If any such usergroup is found, then the associated permission is taken, and, as before, no other permissions are relevant, and all the same remarks apply. Note also that if there is more than one usergroup name which includes the user's name (as, for instance, `EPA???` and `?P?D2?` both include `EPAD25`), then the outcome is unpredictable, since the permission for either usergroup may be taken. It is up to you to avoid putting overlapping usergroups in a permissions list.

4. If a permission has still not been found, then EEP is examined. If it allows any access at all, then EEP applies, and the search stops as before; but if EEP does not allow read or write then the search continues with stage 5.

5. If the file is not a member of a file group then go to step 8.

6. Search the permission list for the parent group, first for the user name, then if that is not found for any usergroup which includes the user name. If neither of these is found examine EEP for the parent group.

7. If a permission has still not been found repeat step 6 for each superior group up to and including the highest group.

8. The 'all files' permissions list is searched for the user name, (ignoring

usergroup names, as at stage 2). If the user's name is found, then the associated permissions are taken and the search stops.

9. If a permission has still not been found, the 'all files' permissions list is searched again for a usergroup which includes the user's name. If such a usergroup is found, the associated permission is taken. All the remarks at stage 3 apply equally to this search.

10. If a permission has still not been found, then EEAFP is examined. If it allows any access at all, then EEAFP applies; otherwise the user is allowed no access.

Restricting access to a file

One point to notice in this procedure is that finding an applicable permission prevents any further searching for other permissions. This has the apparently perverse consequence that granting an access permission can sometimes allow a user *less* access to a file! For example, suppose you own a file called SLUYS and issue the command

```
Command: PERMIT .ALL,E?????,RW
```

Now EDBA12 has RW access to SLUYS (as well as to all your other files).

```
Command:PERMIT SLUYS,.ALL,R
```

Now EDBA12 still has full access to most files, but only R access to SLUYS.

Conversely, removing a permission with PERMIT ...,...,C can have the effect of increasing a user's access to a file.

This feature can be exploited to ban certain users from using a file which is generally accessible to some usergroup or groups. To do this, you use the single letter N ('no access') as the third parameter to PERMIT. Example:

```
Command: PERMIT CHERIMOYA,.ALL,R
Command: PERMIT CHERIMOYA,ERCC04,N
```

This allows all users except ERCC04 to read the file. The trick depends on putting a user's name or usergroup name into one of the 'permissions lists' with no associated permissions at all. You can do it for an individual file or for all files, and for an individual user or for a usergroup - but *not* for 'all users', nor for yourself. If you try it for all users, it has the same effect as C.

3.8 COMMANDS RELATED TO BACKUP STORAGE

3.8.1 The commands CHERISH and HAZARD

```
CHERISH ownfile

HAZARD ownfile
```

In order to provide protection in the event of hardware or software failures in the filestore, some files are periodically copied into a backup store; the frequency of this operation is chosen by the System Manager of the EMAS service. It is normally once each day. All files which are likely to be difficult to reconstruct in the event of file

system corruption should be marked for backing up, by use of the CHERISH command. This command can be used to mark one or more files:

```
Command: CHERISH SNAP
Command: CHERISH ABC.LIST & MINE & COBJECT
```

The command HAZARD is used to remove the CHERISH status.

Notes

* When first created, files are not normally CHERISHed. It is your responsibility to CHERISH your important files.

* Files which are not CHERISHed are deleted automatically if they are left unused for a period (of typically four weeks).

* It is also possible to CHERISH a file group. The affect is to CHERISH all existing members of the group, including members of any inferior groups. Further it will affect future members of the group, and of any inferior groups; they will be CHERISHed automatically when they are created.

* It is less useful to HAZARD a group. It will have the effect of HAZARDing all existing members, and members of inferior groups. It will also ensure that files created in the group, or in any inferior group thereafter will be HAZARDed. But this is what happens by default anyway.

3.9 COMMANDS RELATED TO ARCHIVE STORAGE

The archive store is held on magnetic tape, quite separately from the backup store. It contains files that have been moved there from disc storage as a result of using the ARCHIVE command (see below).

Note that the filestore hierarchy does not extend to the archive store. Files in the archive store retain their full hierarchic filename with full stops in them, but they do not still belong to groups. This means that you can have files in a group called ACCOUNTS in your archive store, even though you no longer have an ACCOUNTS group on-line. In order to restore such a file for use, you would have to create a group ACCOUNTS by using the NEWGROUP command described earlier. Alternatively you could utilize the renaming facility provided in RESTORE (see below).

3.9.1 The command ARCHIVE

```
ARCHIVE file
```

This command, which takes one or more of your own filenames as a parameter, is used to mark files which you want to move from the disc store to the archive store. Note that this command does not take effect immediately: there may be a delay of up to a week before the file is moved. Example:

```
Command: ARCHIVE SOURCE23 & DATA44.V23
```

There are a number of reasons for using this command:

* To dispose of files that are not currently required but may be needed at some later date.

* To clear space in the file index.

* To reduce the charge for keeping files in immediate store.

3.9.2 Obtaining a list of files in the archive store

The command FILES described in Section 3.3 can be used to obtain a list of some or all of your files in the archive store. You must use a type of 'A'. For example:

```
Command: FILES ,A        (all files on archive)
Command: FILES *S,A      (all files on archive with names ending 'S')
```

3.9.3 Moving files from the archive store to the disc store

```
RESTORE file, date, newname
```

The command RESTORE is used to copy a file from the archive store to the disc store. The copy in the archive store is not deleted as a result of this command. The command takes three parameters:

`file` the name of the file being restored

`date` the date of archiving (optional). This parameter is required only when several files with the same name exist on the archive store and the file to be selected is other than the latest. In this case, the date should be typed exactly as it appears in the FILES output, i.e. in the form `dd/mm/yy`.

`newname` (optional) a new name to be given to the file.

The access permissions of the RESTOREd file are reinstated to match those in force when the file was originally archived. Note that these may have been amended by the command APERMIT (see Section 3.9.6) which allows the stored permissions to be modified while the file is archived. The restored file is not CHERISHed.

Examples

```
Command: RESTORE KERN27S
Command: RESTORE IMP907A,23/12/84
Command: RESTORE DATA27,01/02/85,DATA8527
```

If you have access permission to a file in another user's index, you can restore that into its owner's index. You may not use the `newname` facility in this case.

```
Command: RESTORE ERCC19:TOT204X
```

RESTORE will fail immediately if:

* A file of the same name already exists in the index to which the file is being restored.

* The date is typed in incorrect format.

* There is no reference in the archive index to the requested name, or, if a date is used, no reference to a file of the requested name archived on that date.

If the RESTORE command is successfully interpreted then a request is sent to the VOLUMS process to carry out the operation (see Appendix 1). The user can then proceed to give other commands to EMAS. The file is normally recovered from the archive store within 15 minutes.

The RESTORE operation can fail when the VOLUMS process attempts to copy the file

to the disc store. This will occur if:

* There is insufficient room in the file index into which it is to be restored.

* There is a file of the same name in the destination file index. This would only occur if you had created a file of the same name after typing the RESTORE command.

A message from the VOLUMS process will be typed on your interactive terminal, indicating the successful restoration of the file or a reason for failure. A message will also be sent to the owner of the file being restored if that is not the caller. This message will be typed when you next log on if you log off before the file is restored.

3.9.4 The command AWAITRESTORES

```
AWAITRESTORES
```

This command is only of use in Background jobs. If typed during a foreground session it has no effect. Its purpose is to suspend the job until any outstanding RESTORE requests for you have been completed. Thus a job which is intended to compile the source files TRAINS and BUSES, which are held on archive store, could contain the following sequence of commands:

```
Command: RESTORE TRAINS & BUSES
Command: AWAITRESTORES
Command: FORTRAN TRAINS,TRAINY
Command: FORTRAN BUSES,BUSEY
```

Note that AWAITRESTORES will allow the job to continue as soon as the RESTOREs have completed. Completion may not always mean success; as explained above there are some occasions when a RESTORE operation fails.

3.9.5 Destroying files in the Archive Store

```
DISCARD filepattern, datepattern
```

This command is used to delete one or more files in the archive store. The first parameter is a file pattern, which may be simply a filename. The second parameter is a date pattern which may be simply the date.

In both cases the pattern can consist of three kinds of fields:

*	standing for 0 or more characters
?	standing for exactly 1 character, and
other text	standing for itself,

and can have any number of such fields, e.g.

AB*	selects AB, ABC, ABCD, etc.
AB?	selects ABC, but not ABCD
*AB*DE?	selects ABDEF, XABYDEZ, but not ABDEXY
*/10/84	selects all dates in October 1984

A \ appearing at the start of a pattern, negates the sense of the match, so that what matches is not selected, and what does not match is, e.g.

*86	selects all dates not this year

All the candidates are displayed and you are then asked whether you wish to discard

them all, none of them, or be queried about each one individually.

3.9.6 Setting access permissions on archived files

When a file is archived, it retains the individual file permissions that it had when it was on-line. Its permissions can be changed by APERMIT whilst it is still on archive. When a file is restored from archive, it becomes available on-line with those permissions. You can use the RESTORE command to get another user's file restored, provided that you have access permission to the file.

The APERMIT command may be used to extend or restrict access to one of your archived files for other users.

```
APERMIT file, date, user, mode
```

`file` is the name of the file;

`date` the date the file was archived. This can be found by using a call of `FILES filename,A`. If omitted then the most recently archived copy of the file is assumed.

`user` the name of the user (or usergroup) who is to be allowed to use the file.

`mode` the modes of use which will be allowed to that user.

'All files' permissions

Unlike on-line files, it is not possible for you to give another user permission to all of your archived files. However, if you have set a permission for all on-line files, (see Section 3.7.1 above) then that permission also applies to all your archived files.

'All users' permissions

You can allow all users to have access to an individual file. However, you cannot permit all of your archived files to all users. Instead of a user name, you must specify .ALL or omit the user name altogether. For example:

```
Command: APERMIT FRED,17/01/85,.ALL,R
Command: APERMIT FRED,,.ALL,R
Command: APERMIT FRED,,,R
```

Permissions for usergroups

You can specify a usergroup name to allow several users permission to a file. Again, you cannot give permission to all of your archived files to a usergroup name.
A usergroup name is a user name in which some of the characters are replaced by question marks. For example

```
Command: APERMIT .ALL,,ERCO??
Command: APERMIT JOB1,,GRY???
```

Own permission

You cannot alter your own permission for your own archived files. You will always have sufficient permission to restore your own archived files, and you can change your own permission to a file when it is restored. There is no permission setting which will prevent the user from DISCARDing any of his archived files.

Default permissions

If the last parameter of an APERMIT command is omitted, then R permission is assumed. Thus if you wish to permit a file to someone else, then `APERMIT filename,,user` is nearly always sufficient. For example

```
Command: APERMIT RUN1,,ERCC01
Command: APERMIT RUN1,,ER????
```

Multiple permissions

If you call

```
Command: APERMIT FRED,,ERCC01,W    followed by
Command: APERMIT FRED,,ERCC01,R
```

then the file `FRED` is only accessible to `ERCC01` with permission `R` and not `RW` as might be supposed. This is because an APERMIT command supersedes all previous permissions granted for the same file and user. However, this does not prevent a file from accumulating many different permissions for many users and usergroups. For example

```
Command: APERMIT FILE,,ERCC01,R
Command: APERMIT FILE,,ERCC02,RW
```

In this case, `ERCC01` has read permission only, and `ERCC02` has read and write permission.

Revoking permissions

To withdraw the permission on a file, the user must call APERMIT with the letter 'C' (for cancel) as the fourth parameter. When making this call on APERMIT, the user parameter must match exactly the user parameter that was given when the file was permitted. Thus

```
Command: APERMIT FILE1,,ERCC01,R
```

can only be cancelled by

```
Command: APERMIT FILE1,,ERCC01,C
```

and not by

```
Command: APERMIT FILE1,,ERCC??,C
```

If a file is discarded, there is no need to cancel the permissions for it, as they will have been lost with the file.

Restricting permissions

In order to prevent a user, a usergroup or all users from accessing an archived file, the single letter 'N' (no access) is given as the fourth parameter of APERMIT.
For example:

```
Command: APERMIT FRED,01/01/85,ERCC01,N
Command: APERMIT PRIVATE,,ERC???,N
Command: APERMIT PRIVATE,,.ALL,N
```

Summary of parameters for APERMIT

	Set	Revoke	Debar
self	-	-	-
user	file,date,user,perms	file,date,user,C	file,date,user,N
usergroup	file,date,group,perms	file,date,group,C	file,date,group,N
all users	file,date,,perms	file,date,,C	-

Acceptable permissions are W, R
Default permission is R
Permission A means RW

CHAPTER 4
GENERAL FILE MANIPULATION COMMANDS

4.1 INTRODUCTION

Chapter 3 explains the rules for naming files and the commands which operate on files as complete entities. These commands make calls on Director (Appendix 1) which is not concerned with the contents of files and which makes no distinction between the different types of files as understood by the Subsystem.

In this chapter there is a description of the various types of files recognized by the Subsystem and information about how you can use them. There follows a description of some useful commands which provide information about, or alter, the contents of files.

4.2 File Structure

So far as Director is concerned a file consists of one or more pages (of 4096 bytes each) the contents of which are not significant. The Subsystem, in contrast, recognizes the first few (typically 32) bytes of the file as its header. This header contains information about the contents of the file including, for example, its type and the date and time when it was last altered. The Subsystem recognizes the following file types:

Type	Use	Notes
CHARACTER	Holds text - e.g. program source, documentation, commands for OBEY, and DETACH	See Section 13.3
DATA	Holds data for record access or store mapping.	See Section 13.4
OBJECT	Contains compiled program, procedures or commands.	See Section 7.3
DIRECTORY	Contains links between entry names and the names of the OBJECT files which contain the entries.	See Section 7.6
PARTITIONED	Contain members - each of which holds a copy of the contents of a complete file.	See Chapter 20

Table 4.1 Subsystem File Types

4.3 The command ANALYSE

```
ANALYSE file, options, out
```

This command is used to obtain information about a particular file or member of a partitioned file or a group. The parameters are as follows:

`file` The name of the file or member or group to be analysed. It may belong to this user or, if permitted to him, to any other user.

`options` This is used to specify the amount of information required. If it is omitted the minimum information is given, viz. the file's type, size and when it was last altered. If the parameter is given as "*", then all information relevant to the file (or member) type is given.

Alternatively, one or a group of letters can be given. Their meanings are summarized in Table 4.2, together with the file types to which they relate. A letter is ignored if it is not relevant to the type of the file or member specified in the first parameter.

`out` An output device or file; the interactive terminal by default. If an existing file is specified it must be a CHARACTER file.

Examples

```
Command: ANALYSE IMPPROG
Command: ANALYSE PROJECT23B.EXPERIMENT1,*
Command: ANALYSE DIRECTORYAB,PA,.LP10
```

Options	Relevant File Types	Information Given
A	directory	Aliases in the directory file, with the commands for which they are aliases.
E	object or directory	For an object file, the procedure and data Entry names it contains. For a directory file, the Entry names and the corresponding object file names.
F	directory	The names of object Files inserted in this directory.
H	object	The object file History: the name of the source file, (or the component files if LINKed), plus the date compiled (or LINKed).
M	directory or partitioned	For a directory file, the Macros in it, with the filenames containing the macro bodies. For a partitioned file, a summary of information about individual members.
P	all types	Access Permissions. If the file does not belong to you, then your access permission to it. Also the number of users currently using the file.
R	object	The procedure and data References made by the object file.
S	object or directory	Structure: relative start addresses and lengths of areas within an object file. For a directory file, a Summary of its contents.

Table 4.2 ANALYSE Options

4.4 The command COPY

```
COPY file, ownfile
```

This command is used to make a copy of a file. It can be used to copy any type of file, and, subject to suitable access permission having been granted, can be used to make a copy of a file belonging to another user. It takes two obligatory parameters:

`file` the name of the file or member to be copied.

`ownfile` the name of the file or member into which it is to be copied.

Examples

```
Command: COPY KERN27,BACKUP
Command: COPY ERCC27:TEST23,TEST23
Command: COPY PROGRAMAB1,PDXYZ.TEMPORARY1A
Command: COPY ERCC27:FILES.DATA1,D22.DATAX
Command: COPY CONTENTS+CHAP1+CHAP2,BOOK
```

Notes

* See Chapter 20 for details of how to COPY to and from individual members of PARTITIONED files.

* If no file of the given name exists then a new file will be created, with default cherish status and access permissions.

* If the destination file already exists then the command will only succeed if it is of the same type as the input file. Its cherish status and access permissions will not be affected.

* Only character files or data files with identical formats can be concatenated.

* If successful, COPY produces a confirmatory message.

4.4.1 Copying complete File Groups

COPY can also be used to copy a complete file group, including the entire hierarchy of files and groups contained within it. For example if you have stored all your information about a project in a group called `PROJECT86` you could make a copy of it:

```
Command: COPY PROJECT86,OLDPROJECT
```

Note that in this case the command will fail if you already have a group called `OLDPROJECT`.

4.5 The command CONCAT

```
CONCAT
```

This command is used to concatenate (up to 64) character files to create an output character file. It reads the names of the files involved from the primary input device (normally the user's interactive terminal). If it is required to concatenate data files, or a mixture of data and character files, each data file involved should first be converted to a character file (see CONVERT, below).

The filenames should be typed one to a line, the list being terminated with the keyword ".END". This should be followed by a line containing the name of the output file. When input file names are being read from the interactive terminal the prompt is "Conc:". The prompt for the output file name is "File:".

Example

```
Command: CONCAT
Conc:ERCC27:LIST
Conc:MINE.INFORMATION
Conc:FINALINFO
Conc:.END
Outfile:NEWLIST
 3 files concatenated into file: NEWLIST
```

Alternatively, where it is more convenient to permanently store a series of CONCAT instructions, these can be placed in a file, and passed to CONCAT as its controlfile parameter:

```
Command: CONCAT CONFILE
```

Notes

* The files used for input are not altered by this command.

* If the output file already exists it must be a CHARACTER file.

* The output file cannot have the same name as that of any of the input files.

* If successful, CONCAT produces a confirmatory message.

There is also a concatenation operator "+", which can be used when specifying file parameters for the commands COPY, DETACH and LIST, and for the commands which call the compilers: IMP, FORTRAN and PASCAL.

4.6 The command LINK

This command is used in a similar way to CONCAT, but to link object files together. It is described fully in Section 7.3.2.

4.7 The command CONVERT

CONVERT file, ownfile

The command CONVERT is used to convert a data file into a character file. Both parameters are obligatory:

`file` The data file to be converted.

`ownfile` The character file produced from the input file.

Notes

* The output file cannot have the same name as the input file. If a file of the specified new name already exists then its contents are overwritten, but its cherish status and access permissions are preserved. Otherwise a new file will be created, with default cherish status and access permissions.

* If successful, CONVERT produces a confirmatory message.

CHAPTER 5
EDITING CHARACTER FILES

5.1 INTRODUCTION

This chapter describes the two main text editors available on EMAS-3. Sections 5.2 to 5.7 contain a description of the standard Subsystem EMAS editor - EDIT. Sections 5.8 and 5.9 describe the related commands: LOOK and RECALL. Sections 5.10 to 5.19 contain a description of the Edinburgh Compatible Context Editor (ECCE) as implemented on EMAS-3. Sections 5.20 and 5.21 describe the ECCE related commands SHOW and RECAP.

Other text editors may be implemented on EMAS-3. To find out about them type the command `HELP EDIT` or consult the Advisory service.

5.2 THE STANDARD SUBSYSTEM EMAS EDITOR - EDIT

This editor was originally provided for creating and amending program source files. Over the years it has been used with files containing many types of information:

* data for use with programs run on EMAS
* system and user program documentation
* complete jobs for dispatch to remote computers
* text which has no connection with computing, e.g. lecture notes
* messages to be sent via electronic mail

To support this wider use the editor has been enhanced to provide simple text formatting functions and a spelling checker. The formatting facilities are only intended for documents requiring a simple format. They are not intended to replace the facilities of a full text formatting program such as LAYOUT or SCRIBE.

5.3 The command EDIT

This command invokes the standard Subsystem editor, which can be used to examine or alter the contents of a character file: the use of the editor with partitioned files is explained in Chapter 20. There are four ways in which the EDIT command can be used:

* `EDIT newfile` - in this case `newfile` is the name of a file that does not exist. The editor will create a file with the name `newfile` and insert text as instructed by the use of appropriate editor commands. A confirmatory message
  `newfile IS A NEW FILE`
  will be printed.

* `EDIT oldfile` - in this case the file `oldfile` already exists. The effect is to make changes in the file `oldfile` according to the editor commands used.

* `EDIT oldfile, newversion` - in this case the `oldfile` will be copied into `newversion` and will not itself be altered. `oldfile` can be a character member

of a group or a member of a partitioned file. Editor commands used will alter the copy in newversion. Note that if newversion does not exist it will be created, and if it does exist its current contents will be overwritten.

* EDIT oldfile,.NULL - see the command LOOK, described in Section 5.8.

5.4 METHOD OF EDITING

Editing is accomplished by moving a notional cursor through the file and inserting and removing text with respect to the current position of the cursor (hence the name 'context' editor). On entry to the editor the cursor is positioned at the top (beginning) of the file. After each line of edit commands has been typed by the user, one or more lines of the file (depending on the commands given) will be output on the interactive terminal, followed by the prompt Edit:. By this means the user can determine whether his commands have had the desired effect.

5.5 THE EDITOR COMMANDS

All editor commands are single letters (upper or lower case). In some cases they are followed, immediately, by one of the following:

* An integer, which must be typed as a sequence of numeric characters optionally preceded by a minus sign. A single asterisk can be used instead of an integer to mean 'very large positive integer'.

* A text string, which is a sequence of any characters, including newline, delimited by a pair of one of the following characters: / ? ., optionally preceded by a minus sign. For example:

```
/ABC/    ?12*23(A/27)?    .IS
THIS.    -/together/
```

If the delimiter character appears in the text of the string it must be typed twice in order to distinguish it from the closing delimiter. For example:

```
.A=2..3*(B/PI).
```

* A filename enclosed in the characters "<" and ">". For example:

```
<MYFILE>
<ERCC06:EDITTEST>
```

Note that it is permissible to use this form wherever the text string form (above) is allowed, the *complete* contents of the file being then equivalent to the text string. However, where its usefulness is doubtful, the <filename> form is not mentioned in the descriptions of the relevant commands, below. It is included in the summary table (Table 5.1, later in the chapter).

* The single character ' (quote). This is used to specify the same text string as that last given in a use of this editor command. Hence the sequence

```
TM/%END/M1M'
```

is equivalent to

```
TM/%END/M1M/%END/ .
```

Table 5.1 indicates the valid parameter types for each command. Commands can be typed one to a line, or sequentially without separators on a line. For example:

```
TP10M/%ENDOF/
```

Note that a failure in a command within the sequence of commands on a single line will result in the termination of the sequence, at the point of the failure. Spaces within commands, apart from those in text strings, are ignored; hence the following example would have the same effect as the last one:

```
T P 10 M /%ENDOF/
```

Letter case within text strings

All text searching commands are based on case independent text matching. Thus `M/Fred/` will move to `FRED` or `fred` as well as `Fred`. Case dependent matching can be selected by using the Z command. This parameterless command changes the text matching rules between case independent (default) and case dependent. A second `Z` reverts to the original state.

5.5.1 Commands to alter the position of the cursor

The following commands are used to move the cursor around within the file, in preparation for inspecting the contents or altering some part of it. They do not alter the contents of the file.

T - Top. This command takes no parameter. It moves the cursor to the top (beginning) of the file. When text at the beginning of the file is output on the user's terminal, the top of the file itself is represented by `*T*`.

B - Bottom. This command takes no parameter. It moves the cursor to the bottom (end) of the file. When text at the end of the file is output on the user's terminal, the bottom of the file itself is represented by `*B*`.

M - Move. This command can be used with an integer parameter, in which case the effect is to move the cursor from its present position by the number of lines specified by the parameter, and position it at the beginning of the selected line. Thus `M4` means move down the file four lines.

`M-1` means move to the beginning of the previous line.

`M0` means move to the beginning of the 'current line' - the line currently containing the cursor.

If there are insufficient lines in the file the cursor is left at the bottom or top of the file, as appropriate.

Move can also be used with a text string, in which case the cursor is moved from its present position down the file to the beginning of the specified text string. Hence `M/%END/` means move the cursor from its present position to the beginning of the next occurrence of the text `%END`, and `M/4/` means move to the first occurrence of the character `4`. Note the difference between `M4` and `M/4/`. If the text is not found the cursor is left at the bottom of the file.

If the delimited text is preceded by "-", the effect is to move *up* the file to the beginning of the required text. For example:

```
M-/%BEGIN/
```

A - After. This command used with an integer parameter alters the position of the cursor by the number of characters specified by the value of the parameter. Hence A3 means move the cursor forward three characters and A-3 means move the cursor back three characters. All characters in the text are counted, including newline.

If there are insufficient characters in the file to allow the command to complete, the cursor is left at the bottom or top of the file, as appropriate.

The command A can also be used with a text string, in which case it has an effect similar to Move, the difference being that the cursor is moved to after the first occurrence of the specified text. Hence

```
A/%ENDOFPROGRAM/
```

would move the cursor to after the M of %ENDOFPROGRAM. If the text is not found the pointer is left at the bottom of the file. As with Move, the delimited text can be preceded by "-" to cause the search to move up the file from the present position.

G - Go to character. This command is used with an integer parameter and has the effect of moving the cursor to before the character position on the current line specified by the value of the parameter. If the requested position is beyond the end of the current line then the line is extended with space characters up to the cursor. Hence if it is required to put a comment starting at column 40 in an IMP source file, the command G40 would position the cursor correctly for making the insertion. If the value of the parameter is less than 1 it is treated as 1 and if greater than 132 it is treated as 132.

H - Hold. This command is used to move the cursor to the position it was in at the beginning of the last sequence of commands and to undo any editing done by that sequence. This is particularly useful when a mistake is made in typing a text parameter; for example:

```
Edit: M/%EMD/
^*B*
Edit: HM/%END/
^%END
```

5.5.2 Command to insert text

I - Insert. This is used to insert text immediately before the present position of the cursor. When used with a text string, this is the text to be inserted. Hence I/276/ would insert the text 276 immediately before the present position of the cursor. The text string may be of any size and may contain newline characters.

Alternatively "I" can be used with a filename parameter, in which case the complete contents of the specified file are inserted before the current position of the cursor:

```
I<HEADFILE>
```

The file HEADFILE is not altered by this operation.

A member of a group may also be specified:

```
I<GP1.MEMA>
```

A character member of a partitioned file may also be specified.

5.5.3 Commands to delete text

Two commands are provided for deleting text:

D - Delete. This can be used with an integer parameter to delete the number of lines specified by the value of the parameter. For a positive value the lines are deleted from the current line down the file, and for a negative value the lines preceding, but not including, the current line are deleted. Hence:

`D1`	delete the current line
`D3`	delete the current line and the two lines following
`D-7`	delete the seven lines immediately before the current line

If an attempt is made to delete more lines than exist before or after the pointer, the lines that do exist are deleted.

The command D can also be used with a text string, in which case the effect is to delete all the text from the current position of the cursor up to *and including* the first occurrence of the specified text. It is important to note that by default this specified text is case independent. Hence in a textual file the sequence:

```
D/./
```

would have the effect of deleting the whole of the text after the present position of the cursor up to and including the first "." thereafter. As with Move and After, the delimited text can be preceded by "-" to cause the deletion of text to proceed backwards up the file from the present position.

After use of the D command, the cursor is left in the position previously occupied by the deleted text. If the specified text is not found, no text is deleted and the pointer is left at the bottom of the file.

R - Remove. This is used with an integer parameter to remove a specified number of characters, from the present position of the cursor. Hence `R3` means remove the 3 characters immediately after the cursor and `R-10` means remove the 10 characters immediately before the cursor.

If an attempt is made to 'Remove' more characters than exist in the file before or after the cursor (as appropriate) then the characters that do exist are removed.

The command R can also be used with a text string, in which case the effect is to remove the first occurrence of the specified text after the present position of the cursor. Hence `M/Here/R/./` would have the effect of removing only the "." following the first occurrence of `Here`. Note carefully the distinction between `D/./` and `R/./` .

The form `R-/text/` is also permitted. It causes removal of the first occurrence of the specified text when moving back up the file from the present position of the cursor.

After use of R the cursor is left in the position previously occupied by the deleted text. If the specified text is not found the cursor is left at the bottom of the file.

5.5.4 Terminal output from the editor

After each command or each sequence of concatenated commands, the current line (the line containing the cursor) is listed on the terminal. The cursor is normally printed as ^ or ↑, depending on the user's terminal.

In addition the editor command P can be used:

P - Print. This command causes lines to be printed on the terminal. It can take an integer parameter, in which case the effect is to print the specified number of lines; thus `P10` means print 10 lines starting at the current line, and `P-10` means print the 10 lines before the current line and the current line.

P can also take a text string parameter, in which case printing starts at the current line and goes on to the line containing the first occurrence of the specified text.

The form `P-/text/` is also permitted. It causes the printing to start at the line containing the first occurrence encountered of the specified text when moving back up the file from the position of the cursor. In this case printing continues until the position of the cursor is reached.

Note that the cursor is *not moved* by the P command.

5.5.5 The cancel command

If an error is made in typing editor commands the normal rules apply for deleting characters or a whole line (see Section 2.3.5). Additionally, the editor command C can be used:

C - Cancel. This command cancels previous editor commands in the current command string. C must be followed by an integer to specify the number of editor commands before the C to be cancelled. For example:

```
TI/text
here/M27C2I/** text/M20
```

This would have the effect:

```
TI/** text/M20
```

If the value of the integer is greater than the number of commands in the current string then they are all cancelled. It is not possible to cancel the effect of commands in command strings that have already been completed (but see command H, described in Section 5.5.1).

If command repetition is used (see below), it should be noted that when calculating the parameter for C a closing bracket and the integer that follows it are counted as one command and that opening brackets are not counted.

5.5.6 Terminating an edit session

There are two commands for terminating editing:

E - End. E is used with no parameter as the normal exit command. The effect is to return to normal Subsystem command level.

Q - Quit. Q is used as an exit when for some reason the editing done during a

session is not required. The effect is to leave the file being edited in the state that it was in before use of the EDIT command. In order to reduce the risk of a user inadvertently pressing Q and losing his editing unintentionally, this command does not cause an immediate exit but causes the prompt `Quit:` to appear, to which the user should reply `Q` or `Y` if he really does want to exit. Any other reply will result in the edit session continuing.

5.5.7 Preserving editing done so far

The Editor will save any editing work which is in progress when the process is closed down (for example, because of communications breaks or user inactivity), so editing need not be lost. The output will be written into a file :E#EHA which can be inspected and copied to the required destination - or destroyed. The editor will refuse to do any editing while :E#EHA still exists. This feature will also save editing during some system failures, but cannot be guaranteed in the case of a major hardware or power failure.

W - Write. This command can be used at any time during editing to 'Write' all the editing done so far to the output file. This is a useful way to avoid the risk of losing all one's editing if a System failure occurs before one completes an editing session. Note the following:

* The editing done thus far is consolidated into the output file.

* If Q (see above) is used subsequently, the output file will be as it was left by the W, *not* as it was before editing commenced.

* The position of the cursor is not altered by this command.

5.6 MORE ADVANCED FACILITIES IN THE EDITOR

The commands described so far provide most of the commonly required functions of a context editor. There are five further facilities described here which may be of interest to some users:

* repeating commands

* moving a section of text within a file

* extracting part of a file and putting it into another file

* formatting text

* checking the spelling of text.

5.6.1 Command repetition

A single editor command or group of commands can be obeyed repeatedly a specified number of times. The commands are enclosed in parentheses, followed by an integer repetition factor. For example, if it were required to remove all occurrences of the text `REAL` in a file and to replace them with the text `INTEGER`, one could type

```
(R/REAL/I/INTEGER/)1000
```

This assumes that there are not more than 1000 occurrences of the text `REAL`.

An asterisk, "*", can be used in such cases in place of the integer repetition count. It means 'a large number of times' and will cause the command sequence to be executed until a failure occurs, e.g. *B* reached.

Another use of the command repetition facility might be to find out the names of the next 10 subroutines in a FORTRAN program. The command sequence to do this might be:

```
(M/SUBROUTINE/P1M1)10
```

Note that, strictly, this example would print the next 10 lines that contained the text SUBROUTINE. Since this word might appear in a comment the command sequence might not achieve the required effect. Note also the M1 in the command sequence. If this were not included the effect would be to print ten times the next line containing SUBROUTINE. This illustrates the need to consider carefully the effect of repetitive editing commands.

An unsigned integer typed as a command will repeat the previous command string (considered to be enclosed in parentheses) the specified number of times. Thus if

```
D4
```

is followed by

```
5
```

the effect is of

```
(D4)5
```

and a further 20 lines are deleted.

Bracketed commands may be nested.

The normal rules concerning failures within commands are followed. If a failure occurs the whole sequence of commands is aborted.

5.6.2 The separator S

S - Separate. S is used to set a separator *before* the present position of the cursor. It is output as *S* on the interactive terminal. This separator is used to indicate the destination of text being moved (see below). Additionally it has the effect of a separator in the file.

Three commands can be used to move the cursor past the separator set by S:

T - as before, this moves the cursor to the top of the file.

B - as before, this moves the cursor to the bottom of the file.

O - Over. This command, which takes no parameter, moves the cursor to immediately *before* the separator. This is a more efficient operation than, for example, TM*, which would be the form needed if the cursor were positioned after the separator.

All other commands which move the cursor can only move it as far as the separator. This fact can be utilised when it is required to search only part of a file for text strings. Before starting, the user positions the separator at the end of the text to be searched.

The separator can be removed from the file by use of the command K:

K - Kill. K takes no parameter and leaves the cursor at the point previously occupied by the separator.

Note that when S has already been used, the command string KS causes the cursor to be placed immediately after the separator, without changing the position of the separator.

5.6.3 Moving a section of text within a file

The process of moving part of the text of a file from one place to another within the file involves:

* setting a separator at the *destination* of the text, by using the command S
* moving the cursor to the top (start) of the text to be moved
* using the command U to move the text (see below)
* removing the separator with the command K

The commands S and K are described above. The command U is used to move a piece of text within the file:

U - Uproot. U takes either an integer parameter or a text string parameter. When used with an integer the value of the parameter specifies the number of lines to be moved, counted from the current line. When a text string is used the text moved extends from the current position of the cursor up to and including the first occurrence of the specified text.

The form U-/text/ is also permitted. In this case the text moved extends from the first occurrence encountered of the specified text when moving back up the file from the cursor position, down to the cursor position.

In the following example the routine B is to be moved to before routine A.

Initial state of file:

```
*T*
%BEGIN
%ROUTINE A
  text of A
%END
%ROUTINE B
  text of B
%END
  first line of rest of program
    :
    :
%ENDOFPROGRAM
*B*
```

Editing dialogue:

```
Edit:M/%ROUTINE A/S                 /set separator
*S*
^%ROUTINE A
Edit:M/%ROUTINE B/U/%END            /move text, up to and including %END
/:/                                 /followed by a newline
   first line of rest of program
Edit:K                              /clear separator
^%ROUTINE A
```

Final state of file:

```
%BEGIN
%ROUTINE B
  text of B
%END
%ROUTINE A
  text of A
%END
  first line of rest of program
    :
    :
%ENDOFPROGRAM
```

5.6.4 Extracting part of a file

F - File. The command F is used to copy part of a file into another file or to an output device. The parameter with F must be of the form

```
    <filename>
or  <output device>
```

For example: `F<EXTRACT>, F<.LP23>, F<OUT+>`

The abbreviations used for output devices are given in Section 15.6.2. The text which is copied is that which lies between the present position of the cursor and the S separator, if it is positioned lower in the file than the cursor; otherwise, between the cursor and the bottom of the file. The cursor is not moved by this command and the text in the file being edited is not altered. If the parameter specifies a file that already exists, the contents of the file will be overwritten, unless the "file+" form (see example above) is used, when the text is *appended* to the existing contents of the file. If the file does not exist, a new file will be created. In all cases a character file is produced.

F can be used, for example, for extracting one routine from a file for use in another program, or for listing a part of a long file on the line printer. In the previous example the `%ROUTINE B` could be listed on the line printer by using the following sequence of commands:

```
Edit:M/%ROUTINE B/M/%END/M1S      /set separator at bottom of routine B
*S*
^%ROUTINE A
Edit:OM-/%ROUTINE B/F<.LP>        /list routine on line printer
^%ROUTINE B
```

5.6.5 Text layout commands

L - Layout. The command L formats a paragraph of text to the margins specified in the numeric parameter that follows it. It is intended as a simple aid to preparation of minor documents such as MAIL messages and replies.
The paragraph affected is that containing the text pointer and delineated by a blank line (strictly 2 consecutive NL characters) for start and finish; alternatively the markers *T*, *B* and *S* can be used to start or finish paragraphs.
The command lays out the paragraph as defined using a best fit algorithm. It does not attempt to right justify the lines; the command J (justify) should be used for this (see below). The numeric parameter is 100000*P1+1000*Pp+Pe where P1 is the first position on a line to be used, Pp is the position of the first

character of the paragraph (to allow additional indenting of the first line) and Pe is the last allowable position. Thus `L0106080` will lay out a paragraph using printing positions 1-80 with the first line additionally indented to position 6. The parameter can be typed with embedded space to improve readability; thus `L 01 06 080`. The command `L0` will use the previously given parameter to save retyping. The text pointer is left after the double newline terminating the paragraph; thus `(L0106080)6` will format 6 consecutive paragraphs. It should be noted that documents typed in different formats can often be made suitable for L by a single repeated command. A double spaced paragraph can be reduced to single spacing by removing the first of each pair of newlines and after adjustment can be restored to double spacing by duplicating each newline.

J - Justify. The command J operates in the same way as L (Layout) with the additional function of justifying the text after it has been laid out. It does this by inserting additional space characters in each line sufficient to increase the length of the line to that specified in the command. The parameter is as for L; it can be expressed most readily as three numbers which specify the positions of the first character of each line, the first character of the paragraph and the last character of each line. For example the command `J 01 06 080` would layout and justify the current paragraph, indenting the first line to position 6, and using a line length of 80.

5.6.6 The Spelling Checker

The editor incorporates a spelling checker which can be useful when it is used for documentation as against programs. The checker uses two dictionaries. One is maintained by the Documentation Officer and contains more than 30,000 words. The other is maintained for you and contains technical words and private eccentricities. It is held in file :E#EDITDICT. There are three editor commands involved: X, Y and N.

X (No parameter) Move to the next mis-spelt word i.e. the next word that is not in either the system or private dictionaries.

Y (No parameter) Yes it is spelt right. Put it in your private dictionary so that in future it will become an allowed word. After the Y command the cursor is left at the end of the word involved.

N (No parameter) igNore word. The word just found by X is to be ignored and the cursor moved past it. Use this command for proper names or words which you do not wish to store in your dictionary.

Normally you will correct any mis-spelt word as you find it. Note, however, the useful sequence `(XP1N)*` which lists all lines containing mis-spelt words. The spelling checker gets more useful as your private dictionary expands to cover the unusual words you use regularly. There is a limitation in that you cannot remove words from your dictionary, nor can you list your dictionary.

5.7 THE OPERATION OF THE EDITOR

Although it is possible to use the editor with no knowledge of its internal workings, some users might appreciate a brief description. The editor makes use of the virtual memory and handles its files by directly addressing them (see Appendix 2). When editing one file to another it first connects the file in the virtual memory and sets up pointers to its top and bottom. Any text that is inserted is stored in a work area in the virtual memory and each section of text has pointers to its top and bottom.

Any operation which divides a section of text, for example removing a character from the middle of it, results in additional pointers being set up pointing to the beginning and end of the 'hole'. All these pointers are linked together in the logical order in which the sections they point to appear in the file. Note that this order may bear no resemblance to the order in which the sections are laid out in the store. When the edit command E or W is executed, an output file is created, if necessary, and the sections of text are moved into it in the correct order, as determined from the linked list of pointers.

Notes:

* Since the output file is not constructed until the E or W command is executed, all editing may be lost if a System failure occurs during editing (but see Section 5.5.7).

* Since there are pointers to the top and the bottom of the file, moving the cursor to these points with T and B is efficient. When relevant, O is also an efficient command.

* Searching backwards for text by using `M-/text/` or `M-n` (or `A-/text/` or `A-n`) is considerably slower than searching forwards.

Command Summary

Table 5.1 shows the available commands and the parameter types that can be used with each. The final column shows which commands can be used with LOOK and RECALL (see below).

5.8 The command LOOK

The command LOOK is used to activate the editor for the purpose of examining, rather than altering, a file. It takes one parameter - the name of the file to be examined - with a default of T#LIST, the default compiler listing file (see Section 7.2.1).

A similar effect can be achieved by typing

```
EDIT filename,.NULL
```

There are three differences between using LOOK and EDIT:

* The editor commands D, G, I, J, L, R, U and W are not allowed.

* The prompt at editor command level is `Look:` rather than `Edit:`.

* The parameter for LOOK has a default of T#LIST.

Otherwise the facilities available are identical.

Command	Use	Valid Parameters						
		None	Integer	Text string	- Text string	Quote	Filename	Allowed in LOOK and RECALL
A	After		*	*	*	*	*	*
B	Bottom	*						*
C	Cancel		*					*
D	Delete		*	*	*	*	*	
E	Exit	*						*
F	File						*	*
G	Go to		*					
H	Hold	*						*
I	Insert			*		*	*	
J	Justify		*					
K	Kill	*						*
L	Layout		*					
M	Move		*	*	*	*	*	*
N	igNore	*						*
O	Over	*						*
P	Print		*	*	*	*	*	*
Q	Quit	*						*
R	Remove		*	*	*	*	*	
S	Separate	*						*
T	Top	*						*
U	Uproot		*	*	*	*	*	
W	Write	*						
X	Spell	*						*
Y	Yes	*						*
Z	Case	*						*

Table 5.1 Summary of Standard Subsystem Editor Commands

5.9 The command RECALL

This command is used to interrogate a file containing a copy of all your recent interactive terminal Input/Output operations. The RECALL keyword of command OPTION must have been set to either PERM or TEMP (see Section 17.9) to specify that the terminal monitoring facility is required. RECALL takes no parameter. There are three differences between RECALL and EDIT:

* The editor commands D, G, I, J, L, R, U and W are not allowed.

* On entry the cursor is at the bottom of the file - i.e. pointing at the end of the most recent information.

* The prompt at editor command level is `Recall:` rather than `Edit:`.

5.10 THE EDINBURGH COMPATIBLE CONTEXT EDITOR

The Edinburgh Compatible Context Editor (ECCE) was developed originally by members of staff of the Department of Computer Science, University of Edinburgh. The EMAS version has been enhanced by ERCC staff, in response to feedback from users. Apart from extensions to the command syntax there are the following major extensions:

* ECCE now incorporates a spelling checker.

* There is a command to enable you to format your text, with or without right hand justification.

* If the file is an IMP program you can check the correctness of its syntax without leaving the editor.

5.11 CALLING THE EDITOR

The editor can be used to manipulate any character file: the use of the editor with partitioned files is explained in Chapter 20. There are three ways in which the ECCE command can be used:

1) `Command: ECCE oldfile` to amend an existing file; a warning is given if the file does not exist.

2) `Command: ECCE oldfile, newfile` to create (or overwrite) `newfile` with the edited `oldfile`, leaving `oldfile` unaltered.

3) `Command: ECCE ,newfile` to create (or overwrite) `newfile`

In cases (2) and (3) the user is warned and given a chance to exit before an existing file is overwritten.

5.12 USING ECCE

After invoking ECCE, the user issues commands from his terminal, one to a line or many to a line, and in either upper or lower case. Spaces have no significance except within text strings, and command lines are terminated either by a newline character or by a semicolon (;). If semicolons are used, one physical line can contain

more than one command line. After reading the command line the editor checks it for syntactic correctness and then executes the individual commands in left to right order. A syntax error in a command line (such as an unmatched string delimiter) causes the whole command line to be ignored and an error report to be produced (see Section 5.12.7).

To terminate the editing session and close all input and output files, the command %C is issued. In this case %C must be the only command in the command line. If you decide to abandon an editing session you can exit by using the command %Q. You will be asked to confirm that you really want to exit from the editor, losing all the changes you have made.

5.12.1 The file pointer

Editing commands (except the "special" commands such as %C) operate on the source text at the "current position" in the text. In order to define the current position, the editor maintains a pointer, henceforth called the "file pointer". Conceptually this is always between two characters of the source text, or immediately before the first character of the file or immediately after the last character in the file. When printing out a line on the user's terminal the editor identifies the file pointer by typing an up-arrow or caret character in the appropriate position unless this is at the beginning of a line.

Example:

```
This is an up-ar^row or caret character.
But here the file pointer is at the start of the line.
```

For convenience, the end of the file is treated as an extra line following the last line in the file. This extra line is typed out as **end**. For example:

```
This is the last line of my file.
**end**
```

By default ECCE types out the current line (i.e. the line containing the file pointer) after the execution of each command line, unless the line was the line most recently typed out and the file pointer was not moved (forwards or backwards) across a line boundary by the latest command line. Thus, in all cases the last line printed on the user's terminal is the line containing the file pointer.

5.12.2 A simple subset of commands

In this section a simple subset of ECCE's facilities will be described. This is sufficient to perform most editing operations. More sophisticated commands and programmed commands will be described later. The commands described allow the file pointer to be moved forwards and backwards in the source text, a line of source text killed (removed), a line of text inserted into the file, occurrences of a specified text string found, lines of text printed, and the editing session closed.

All these commands (except %C) can be followed by a positive integer, zero (0), or *, to specify how many times the command is to be repeated. Zero and * denote indefinite repetition, i.e. the command is executed repeatedly until it fails or until a large number of repetitions (currently 5000) have been performed.

%C — Close the editing session. The input and output files are closed and control returned to command level. %C must be the only command in the command line.

M — Move the file pointer to the start of the next line. This fails if the file pointer is already at the end of the file (that is, after the last line of the file).

M- — Move the file pointer back to the start of the previous line. This fails if the current line is the first line of the file.

K — Kill the current line. That is, remove the whole of the current line from the file. The file pointer is left at the beginning of the next line (as for M). This command fails if the file pointer is already at the end of the file.

G — Get a line of text from the user's terminal and insert it between the end of the previous line (if any) and the *start* of the current line. The file pointer is left at the start of the current line (that is, immediately after the new text). The user is prompted for input with a colon (:) and the command will fail if the first character of the input line is a colon, in which case the file pointer is not moved. The rest of the line after the colon (if present) is treated as a command line. The form `G*` or `G0` can be used when typing in an indeterminate number of lines; see the example below.

F/TEXT/ — Find TEXT. Search the rest of the file, starting at the file pointer, for the first occurrence of the string TEXT. A string is any sequence of characters, *excluding* newline, enclosed in quotes. Any symbol other than those which have a defined significance in the command language may be chosen to represent the quotation mark, for example / ' " # or . . The chosen quotation mark must not occur within TEXT. The file pointer is left immediately before the first occurrence of TEXT if TEXT is found. Otherwise the command fails and the file pointer is left at the end of the file. If the command is repeated, then the occurrence of TEXT just found is ignored in the subsequent search for TEXT (see Section 5.12.5).

P — Print the current line at the user's terminal. If the multiple form of P is used, for example `P6` or `P*`, then a move (M) is performed after the first and subsequent Ps (but not after the final one). In this way a sequence of lines may conveniently be printed at the user's terminal. The simple command P can never fail and does not move the file pointer. The multiple form of the Print command (for example `P10`) fails only if an implicit attempt is made to move beyond the end of the file (for example, trying to print 10 lines when the current line is only 8 lines from the end of the file), and always leaves the file pointer at the start of the last line printed, or at the end of the file.

The P command identifies the file pointer by typing a caret or up-arrow character in the appropriate position, unless this is at the start of a line. The end of the file is identified as `**end**`.

It should be noted that the multiple form of P is not strictly repetition, for the effect of P followed by P is to print the current line twice, whereas the effect of P2 is to print the current line and the next line, and move the file pointer to the start of the next line. Generally this does not cause any problems or confusion.

5.12.3 An example of the use of ECCE

In this section ECCE is used interactively to create and update a file of text, which will contain part of a poem by Roger McGough called "Discretion". The complete dialogue between ECCE and the user is displayed. The user is prompted for command lines with a ">" character and for input with a ":". Starred lines are

ECCE's normal (default) monitoring output (see Section 5.12.11). Note that "*" is not printed at the user's terminal, and that ">" and ":" are not inserted into the source text. Explanatory comments occur in the right hand half of the line starting with a "/" character.

```
Command: ECCE ,POEM              /invoke the editor to make a new file
Edit
>g*                              /get some lines of input
:Discretion
:Discretion is the better part of Valerie
:(though all of her is nice)
:lips as warm as strawberries
:eyws as cold as ice
:the very best of everything
:only will suffice
:not for her potatoes
:and pudding made of rice
:
:Not for hwr potatoes
:and puddings made of rice
:she takes carbohydrates like God takes advice
::                               /end the input for now

>m-9                             /move back 9 lines
* eyws as cold as ice
>k g                             /kill it and get a replacement
:eyes as cold as ice
>f/pudd/                         /missed the "s" off the end
* and ^pudding made of rice
>k                               /delete the line
*                                /current line after deletion (blank)
>g                               /get a replacement
:and puddings made of rice
>f/hwr/
* not for ^hwr potatoes
>k g                             /kill it, and replace it
:not for her potatoes
>m*                              /move to end of file
* **end**
>g*                              /and continue the input ...
: ...
```

5.12.4 On-line HELP for ECCE

If you need help with understanding ECCE whilst you are using it you can use the special command %H. This takes you into the VIEW system at the start of the section concerning ECCE. See Section 2.7.6 for more information about using VIEW. When you have finished reading the HELP information type 'Q' to the prompt 'View:' and you will be returned to ECCE.

5.12.5 Text location and manipulation commands

It is apparent from the above example that the use of only the very limited subset of commands introduced so far is clumsy, tedious, and error prone. The commands described below can be used to move the file pointer to immediately before or after a specified text string, and to insert, substitute, or delete specified strings of text, rather than whole lines.

In many of the commands introduced in this section a search for a text string is implied. This search always begins at the character immediately following the file pointer and fails, with the exception of Find, if the end of the line is encountered before the text is found. Find fails only at the end of the file.

In successive searches for a text string by Find or Uncover, the occurrence just found is ignored. By contrast, the Delete command *will* operate on the occurrence of text just found.

In all the following examples the "/" character is used as a string delimiter. Any character with no other significance to the editor could be used, for example ' or # or $ or . (or / itself).

Note the effect on the text location commands of %U and %L (described in 5.12.12).

F/TEXT/ Find TEXT. Search the rest of the file, starting at the file pointer, for the first occurrence of TEXT. The file pointer is left immediately before the first occurrence of TEXT if it is found. Otherwise the command fails and the file pointer is left at the end of the file. If the previous command was a text location command (F, U or V, see below) then an occurrence of TEXT just located is ignored when searching for TEXT. This applies particularly to the case of Fs being repeated; for example, F/TEXT/5 finds the 5th occurrence of TEXT after the current position of the file pointer.

S/STR/ Substitute STR for TEXT. If the previous command was F/TEXT/, U/TEXT/ or V/TEXT/ (see below) then delete TEXT and insert STR. The file pointer is left immediately to the right of STR. The command fails if the previous command was not an F, U or V. Note that STR may be the null string, so that F/ABCD/S// deletes the first occurrence of ABCD.

T/TEXT/ Traverse TEXT. Search the current line, starting at the file pointer, for the first occurrence of TEXT and move the file pointer to immediately after it. If TEXT is not found on the current line the command fails; in this case the file pointer is not moved. Traverse is rather like Find except that the file pointer is left *after* the occurrence of TEXT, but note that it is *not* a text location command (Substitute cannot be used after it). Traverse is useful in many circumstances: for example, when adding an "s" (or anything else) to the end of a word.

D/TEXT/ Delete TEXT. Search the current line, starting at the file pointer, for the first occurrence of TEXT, and delete it. The command fails if TEXT is not found before the end of the current line. If the command succeeds then the file pointer is moved to the position previously occupied by TEXT; otherwise the file pointer is not moved.

I/TEXT/ Insert the specified TEXT immediately before the file pointer. The file pointer is left immediately after the inserted TEXT. Insert will fail if the line length is already greater than 160 characters. If Insert fails then the file pointer is not moved.

U/TEXT/ Uncover TEXT. Search the current line, starting at the file pointer, for the first occurrence of TEXT and remove all characters between the file pointer and the start of TEXT (TEXT itself is *not* removed). The file pointer is left immediately to the left of TEXT. If TEXT is not found on the current line then the command fails and the file pointer is not moved. If the previous command was a text location command (F, V or U) then an occurrence of TEXT just located is ignored. If TEXT is found, it may then be replaced by using the S/STR/ (Substitute) command.

V/TEXT/ Verify that TEXT occurs immediately to the right of the file pointer: fail otherwise. If successful, S/STR/ can be used to substitute STR for the TEXT just verified (see above for S). This command is of use in programmed commands (see Section 5.13).

The following combinations of commands are often found to be useful:

```
F/TEXT1/ S/TEXT2/     /applies to rest of file
D/TEXT1/ I/TEXT2/     /applies to rest of line
T/word/ I/ending/     /applies to rest of line
U/./                  /delete rest of sentence (strictly, up to next full stop)
U/./S/,/              /delete rest of sentence and
                      / change full stop to comma
```

5.12.6 A further example of the use of ECCE

In the earlier example of the use of ECCE only a very basic subset of commands was used. In this section the same example is reworked to show how the text location and manipulation commands, just introduced, can be used to make the necessary changes more easily.

The comment and monitoring conventions, and the command and input prompts, are exactly the same as in the previous example.

```
Command: ECCE ,POEM              /invoke the editor to make a new file
Edit
>g*                              /get some lines of input
:Discretion
:Discretion is the better part of Valerie
:(though all of her is nice)
:lips as warm as strawberries
:eyws as cold as ice
:the very best of everything
:only will suffice
:not for her potatoes
:and pudding made of rice
:
:Not for hwr potatoes
:and puddings made of rice
:she takes carbohydrates like God takes advice
:a surfeit of ambition
:is her particylar vice
:Valerie fondles lovers
:like a mousetrap fonles mice
::                               /end the input for now
```

```
  >m-0                                  /move back to start
* Discretion
  >f/eyws/ s/eyes/                      /correct first mistake
* eyes^ as cold as ice
  >f/pudding/t/g/                       /missed the "s" off the end
* and pudding^ made of rice
  >i.s.                                 /so put the "s" back in
  >f/hwr/ d/w/                          /remove the mistake
* not for h^r potatoes
  >i/e/                                 /and correct it
  >f/y/                                 /look for next blunder
* she takes carboh^ydrates like God takes advice
                                        /not there yet!
  >f/y/                                 /so repeat the command
* is her partic^ylar vice
  >d/y/ i/u/                            /fix the error
                                        /N.B.  D is not a text location
                                        / command
                                        /so "y" just Found is deleted
  >f/nle/ s/ndle/
* like a mousetrap fondle^s mice
  >%c                                   /end the edit

  Command:                              /next command prompt
```

5.12.7 Command failure

A command can fail in either of two ways. Firstly it may be syntactically incorrect, in which case the whole command line is ignored and an error message is produced. Secondly it may fail in execution, in which case a failure message is produced, the current line (at the time of the failure) is printed, and the rest of the command line is ignored.

In general the failure of a simple command leaves the file pointer unmoved; however, if the failing command is part of a command line or repeated command then the file pointer is left positioned by the last successfully executed command.

Syntactic errors include commands that are not recognized, mismatched string delimiters, mismatched parentheses (see Section 5.13), command line size exceeded (currently the limit is 100 simple commands), and so forth. For example:

```
      >[
       [?                       /unknown command
      >f/hello.
       TEXT FOR F?              /mismatched string delimiters
      >g/abc/
      G /?                      /wrong syntax entirely
```

but

```
      >[p2;m                    /[ (and rest of command line) is ignored
                                / but M (next command line) is executed
```

Examples of execution failure:

```
      **end**                   /at end of file
      >m                        /can't, so it will fail
      Failure: M
      **end**
```

and

```
>m                      /move to next line
How now brown cow.
>s/horse/ m             /meaningless - S can only follow F, U or V
Failure: S'horse'
How now brown cow.      /note, Move not executed
```

or

```
>p                      /print current line
How now brown cow.
>t/now/2                /note that "now" occurs only once
Failure: T'now'
How now^ brown cow.     /second T/now/ not possible
```

or

```
>g3                     /get 3 lines
::                      /didn't mean it, so get out of G
Failure: G
... current line ...    /echo of current line on error
```

Note that indefinitely repeated commands such as g* or m* produce no failure reports, but that the failure condition is used to terminate the repetition.

5.12.8 Character manipulation commands

Of the commands met so far, M and M- move the file pointer past a whole line and K deletes a whole line. The next five commands operate on individual characters within a line.

R — Right shift the file pointer by one character position. This fails if the file pointer is already at the end of the current line.

L — Left shift the file pointer by one character position. This fails if the file pointer is already at the beginning of the current line.

E — Erase the character immediately to the right of the file pointer. This fails if the file pointer is already at the end of the current line.

E- — Erase the character immediately to the left of the file pointer. This fails if the file pointer is already at the beginning of the current line.

C — Causes Case inversion of the character to the right of the file pointer, then shifts the pointer one position right. Hence command C2 causes ^Ab to become aB^. If the character is not a letter it is left unaltered but the shift right still takes place. The C command fails if the file pointer is already at the end of the current line.

C- — Causes Case inversion of the character to the left of the file pointer, then shifts the pointer one position left. Hence command C-2 causes cD^ to become ^Cd. If the character is not a letter it is left unaltered but the shift left still takes place. The C- command fails if the file pointer is already at the start of the current line.

H — Move file pointer Horizontally to the position specified on the current line, inserting space characters if the position is beyond the end of the current text. For example H12 means move the file pointer to just before column 12. You can use this command to move the pointer left or right, depending on where it is initially. Note that H0 and H mean the same as H1, the first column. This command is unusual in that it does not take a repetition count. The number following it specifies the column.

It is useful to note the effect of indefinite repetition (*) with six of these commands:

```
R*      /move the file pointer to the end of the current line.
L*      /move the file pointer to the start of the current line.
E*      /erase from the file pointer to the end of the current line.
E-*     /erase from the start of the current line
C*      /invert the case of the text
        / from the file pointer to the end of the current line.
C-*     /invert the case of the text
        / from the start of the current line to the file pointer.
```

5.12.9 Breaking and joining lines

The following commands are used to break a line into two parts, and to join two lines together.

B Break the current line into two parts at the file pointer. That is, insert a newline character immediately to the left of the file pointer, so that the second part becomes the new current line. B never fails. Breaking a line at its beginning (that is, when the file pointer is at the start of the line) has the effect of inserting an empty line immediately before it (which may also be achieved by means of G, then replying carriage return to the G prompt).

J Join the current line and the next line by removing the newline character at the end of the current line. This command fails if the current line already exceeds 120 characters in length. If the command succeeds then the file pointer is left at the join. Otherwise the file pointer is not moved.

5.12.10 The repetition command

A command line consisting solely of a number or * causes the previous command line to be executed the specified number of times. For this purpose repetition commands do not count as command lines, so that it is the most recent non-repetition-command command line that is repeated. Zero (0) or * causes the previous command line to be executed until failure. For example:

```
>M                  /move to the next line
>3                  /move 3 more times (if possible)
>2                  /move twice more (not 6 times)
>m-2                /move back two lines
>4                  /now back 8 more (not 4)
>*                  /all the way to the start of the file
>P3                 /print the first 3 lines of the file
The cat
sat on
the mat.  Clever cat.
>                   /ready for next command
```

Note that this command sequence was executed in terse mode (see Section 5.12.11) and that ">" is ECCE's command prompt.

5.12.11 Monitoring commands

There are three special commands that are used to change the amount of "echoing" that ECCE performs. These have no effect on the source text. Special commands must appear as the only command in the command line.

%M Monitor normally. This is the default when ECCE is first invoked (see also Section 5.15). The current line of the source text is typed at the user's terminal after the execution of each command line unless two conditions hold true. First the current line is the latest line to have been typed out, and secondly the current command line did not cause the file pointer to be moved across a line boundary at any stage in its execution. The current line is not re-echoed if the last command in the latest command line was a Print command.

%F Full monitoring is turned on. The current line is typed out after the execution of each command line, unless the last command executed was a print (P) command. Thus at least one line is output on the user's terminal following each command line.

%T Terse mode. Turn off all monitoring. The current line is typed out only if explicitly requested by means of the print (P) command, or if the command line fails.

5.12.12 Upper and lower case control commands

There are two special commands which affect the operation of the text location and verification commands (i.e. D, F, T, U and V). As before, a special command must appear as the only command of the command line.

%L Lower case. With %L selected, the case of the string being sought *is* significant and must be matched exactly. %L is the initial default (see also Section 5.15).

%U Upper case. The user is considered to be at an upper case terminal, and so cannot distinguish upper case text from lower case text in the file being edited. For this reason, the case of the text scanned by a text location or verification command is not regarded as significant; thus, for example, `F/ABC/` and `F/abc/` are treated as entirely equivalent, and either would match "abc", or "ABC", or "Abc", or "AbC", etc., in the file. %U must be specified if required.

5.12.13 Dealing with non-printable control characters

If you need to insert non-printable control characters in a file, e.g. newpage characters then you should type them using their hexadecimal codes, delimited by '&'. For example the hexadecimal code for ESCape is '1B', so to insert an escape character in a file you should type:

```
>I&1B&
```

Note that normally each character is written as two hexadecimal digits even when the first digit is zero. You may, however, omit the leading zero of the first digit in a string if you wish. For example the hexadecimal code for the Newpage (Form Feed) character is '0C'. If you wish to locate two consecutive newpage characters in a file you could type:

```
>F&C0C&
```

Appendix 3 contains a list of characters and their hexadecimal codes.

When ECCE prints lines containing control characters it normally prints the whole line, with no special treatment for the control characters. This can have odd effects. For example if the line contains a backspace character then it will cause the preceding character to be overwritten on the screen. A special command %D is

available to control the way in which ECCE prints control characters. You can assign a value between 0 and 2 to %D, e.g.:

```
>%D=2
```

The effect will be as follows:

If %D=0 is in force ECCE prints the whole line, including control characters. (This is the default).

If %D=1 is in force ECCE prints a '?' at the position of the control character.

If %D=2 is in force ECCE prints a '?' at the position of the control character and also rings the terminal's bell (if it has one).

If you just type '%D' with no parameter you will get an explanation of these alternatives. The %D value can be saved for future sessions with ECCE, by use of the %P command (see also Section 5.15).

5.12.14 Context specification - D, F, T and U revisited

So far D, F, T and U have been presented with their default contexts only. However, the number of lines to be searched in a text location command may be specified explicitly by inserting a number between the command letter and the text. This makes it possible to limit the scope of Find (by default the rest of the file), and extend the scope of Delete, Traverse and Uncover (by default the rest of the current line). For example:

```
D50/DOG/        /search 50 lines at most
F10/HELP/       /search 10 lines at most
T3/END/         /search 3 lines
U*/./           /search rest of file
```

The effect of specifying a context is best explained with reference to a specific command, for example, `D50/DOG/`. This operates in the following way:

1) Search the current line for `DOG`.
2) If found then Delete `DOG` and terminate the command successfully.
3) Has step 1 been executed 50 times?
4) If so then command fails.
5) Otherwise Move to next line and go back to step 1.

If this command succeeds the effect is to delete the first occurrence of `DOG` within the next 50 lines (current line + next 49 lines). If it fails the effect is `M49` (*not* `M50`).

The other examples above have the following effects respectively: Find the first occurrence of `HELP` within the next 10 lines (or `M9` if `HELP` doesn't occur within the next 10 lines), Traverse the first occurrence of `END` within the next 3 lines (or `M2` if `END` does not occur within the next 3 lines), and more dangerously, Uncover all text up to the next "." (or Kill the rest of the file if "." does not occur before the end of the file). Note that this final command `U*/./` is very dangerous, as all the text examined is Killed.

These commands may also be repeated. For example:

```
F15/HELP/2
```

This command seeks the string `HELP` within the rest of the current line just found and seeks `HELP` again within the rest of the *new* current line and the fourteen lines

following it. The command will fail if any repetition of F15/HELP/ fails, in which case the effect is either M14 if the first search fails, or M(between 14 and 28) if the first succeeds and the second fails. If the command succeeds the file pointer is left immediately before the second occurrence of HELP.

5.13 PROGRAMMED COMMANDS

It will have been noticed that several commands may be put on one command line to create a command "program". Furthermore, command lines may be repeated a specified number of times, or until failure. This section describes how more general "programs" can be written.

( ) Bracketing. A string of commands bracketed together is treated as one command for purposes of failure and repetition. For example:

```
(M I/    /)*
```

inserts four blanks at the start of each line of the rest of the file.

Brackets may be nested.

? Optional execution. Any command failure condition is ignored. For example:

```
D/PHONE/?
```

deletes the word PHONE from the current line if it occurs on the current line. Otherwise it does nothing; in particular it does not fail.

\ Inverted success/failure condition. A command followed by \ instead of a repetition number has its failure condition inverted, i.e. it succeeds if and only if it fails! For example:

```
(MV/%R/\)0
```

This programmed command moves the file pointer to the start of the next line then repeats if V/%R/ fails. That is, the move is repeated until a line beginning with %R is found or until end of file is reached.

, (comma) Alternative command sequences. Sequences of commands separated by commas form alternatives. If the first alternative fails the next is tried, and so on. If an alternative succeeds then the remaining alternatives are ignored. For example:

```
(D/CAT/,D/DOG/,M)
```

This command either deletes CAT or DOG on the current line, or moves the file pointer to the start of the next line. (Try to Delete CAT. IF unsuccessful THEN try to Delete DOG. IF still unsuccessful THEN try to Move to next line.) Note too that command lines may be broken immediately after a comma and thus span two or more physical lines. For example:

```
D/CAT/,
D/DOG/,M
```

The file pointer can be moved by an unsuccessful attempt to execute a command alternative, and this may well affect the execution of subsequent alternatives. One must therefore be very careful when using commands with alternatives. For example:

```
MD/CAT/,MD/DOG/
```

would not have anything like the effect of the previous example. In this case we move to the next line, and then if D/CAT/ fails we move to the next line and try D/DOG/. If CAT and DOG occurred on consecutive lines and the command were started on a line containing CAT (next line contains DOG) then *no* occurrences of CAT or DOG would be deleted.

By judicious use of ( ) , ? \ and repetition, quite complex editing sequences can be programmed. For example:

```
(d/cat/ i/chat/ 1* , d/dog/ i/chien/ 1* , m)*
```

This "program" translates the words "cat" and "dog" to their French equivalents (but note that "catalogue" gets translated to "chatalogue"!). Note also the following useful commands:

```
(RM)0                 /find the next empty line
                      / (assuming no lines consist only of space characters)
(R0(LD/ /)0M)0        /remove trailing spaces from all lines
((I/     /M)60B6)*    /right shift all lines by four spaces
                      / and separate pages with 6 empty lines
```

This final command is useful for paginating a file before printing it on a line printer, but note the assumption of 66 lines per page, and no carriage return characters in the file.

Note that "*" and "0" (indefinite repetition) are entirely equivalent.

" Can be used instead of a delimited string. Refers to the last string specified for the same command. For example:

```
>f/Fred/ s/Joe/ m3 f" s"
```

Note that following Q (query spelling) F" can be used to find subsequent occurrences of the word involved (see Section 5.19.2).

^ Can be used in a command line. Two occurrences delimit text within a line of the primary or secondary file, as defined by the current position when ^ is specified. Thus:

```
Here^ is a short piece of text

>f/short/^r0^
```

would delimit the text 'short piece of text'. Delimited text can subsequently be referenced by use of @ instead of a normal string. Thus:

```
>f@
```

following the above command line would find the next occurrence of the string 'short piece of text'.

- The current position must stay on the same file line while the two ^s are given, and the file must not be altered during the process. The start and finish positions can be specified in either order. Further examples:

```
>m-0(^r0^bi@m)0
```

causes every line in the file to be repeated

```
>^(rv/ /\)*^(pmf@)*
```

marks the text from the current position to the next space (or end of line), and then locates and prints all subsequent lines containing occurrences of it.

```
>LOP
Dec 17 1945
>^R4^E-4R3I@P
17 Dec 1945
```

Useful for moving columns within a table. In this case the day and month have been exchanged.

5.14 MACROS

Four "macro" commands can be defined by the user, namely W, X, Y, and Z. When invoked, the effect of a macro is *exactly* as if the macro text had been typed instead of the macro letter. For example:

```
>%X=(RM)*          /note: %X=macro text, not X=macro text
```

This defines X to be the next-empty-line command (see above). The effect of X (or x) in a command line is then exactly the same as (RM)*. Macro text length is restricted to 127 characters. In addition, both macro texts and command lines may contain no more than 100 command units, where each comma, bracket, \, number, and simple command counts as one unit.

For example, if X were to be defined as MK, then

X4 is exactly equivalent to

MK4 and not to

(MK)4 as might be imagined. But note that

(X)4 *is* equivalent to

(MK)4

A macro can be defined as several command lines separated by semicolons. For example:

```
%X=f1/TEXT1/;f1/TEXT2/
```

```
%X=%s;f/%ROUTINE/nf/%END/ma;%s
```

The latter example involves secondary input, which is described in Section 5.17.

5.14.1 Extended macros

An alternative method is provided for carrying out complicated edit sequences repeatedly. To use it you should create a file which contains the ECCE commands to be obeyed. To execute the commands in the file you should type the special command %I, as follows

```
%I=file
```

where `file` is the name of the file containing the commands.

If the last command in the file is a %C then the ECCE command will end at that point. Otherwise a return will be made to the '>' prompt for further keyboard input.

Apart from using a file of commands in this way you can also create a file of commands to be obeyed automatically every time you start to use ECCE. This file should be in your current USEGROUP (Section 3.2.4) or not in any group, and should have the name ECCE#START. The same facility is available for SHOW and RECAP (Sections 5.20 and 5.21) using files called SHOW#START and RECAP#START respectively.

This facility can be used in conjunction with the wild-card facility in the EMAS command language to allow you to carry out an operation on a whole group of files. For example if you want to check the use the COMMON statement in a number of related Fortran source files you could proceed as follows.

First create a file, called SHOW#START which should contain the text:

```
(F/COMMON /PM)*
%C
```

Now call SHOW by using a wild-card parameter. For example if the files had names such as `PACK01S`, `PACK02S` etc. then the command would be:

```
SHOW PACK*S
```

The effect would be to make a call on SHOW with each file with a matching name, printing all the lines which contained the text `'COMMON '`.

5.15 PRESERVING YOUR OPTIONS

The special command %P (save Profile) can be used at any time to preserve all the optional settings and the W, X, Y and Z macro texts. This includes the %A (Adjust) parameters (see Section 5.19.1), the monitoring level (%M, %F or %T), the %D setting and the setting of case significance for searching (%U or %L).

The effect of saving this profile is that all these values will be set next time you use ECCE.

5.16 RECOVERING FROM ERRORS

If you make a mistake whilst you are using ECCE you can retrace your steps in various ways depending upon the severity of the mistake.

* If you notice the mistake before you press <RETURN> at the end of the command line then you can correct it in the usual way by using the DEL key.

* If you notice a mistake after you have typed a command line you can type %^ to restore the position as it was before you typed the line, regardless of whether it altered the file. %^ can be used repeatedly.

* If you notice a mistake later you can use the command %R to Retrieve editing. This takes you back to before the last command line which altered the file. %R can be used repeatedly.

* If the mistake is more complicated you can exit from the editor by using the special command %Q. This will leave the primary output file unchanged, at the cost of losing all the editing you have done in this session. To avoid the danger of you losing all your editing the %Q command is followed by a request for you to confirm that you really do want to abandon this editing.

In order to protect your editing in the event of your interactive terminal being disconnected whilst you are using ECCE (e.g. because of a communications problem) there is a mechanism to store the current state of the edited file in a file called ECCE#BACKUP when such a break occurs. You can also use this mechanism yourself by using the special command %B. This writes the edited version into the file ECCE#BACKUP. This gives you a place to go back to if you are about to embark on some particularly complex edit sequence. If this sequence goes wrong you should exit from ECCE by using %Q, and then rename ECCE#BACKUP to some other name. It will contain the state of the edited file at the time when you typed %B.

Note that the %C (Close) command, which is the normal command to end an edit session, destroys the file ECCE#BACKUP if it exists. If ECCE#BACKUP exists when you call ECCE, it will print out a warning message.

5.17 SECONDARY INPUT

Secondary input is a feature that allows parts of a second input file to be copied into the main file. The secondary input file cannot be changed, and any file changing commands (I, U, E etc.) used in secondary input mode will fail.

To enter secondary input mode the command

```
%S=filename
```

is issued. This causes ECCE to prompt for commands with ">>" rather than ">", and the editing commands thenceforth operate on the file specified (the "secondary input file") rather than on the primary file.

To leave secondary input mode a `%S` command without the `"=filename"` part is issued and ECCE then prompts for commands with ">" again.

Subsequent `%S` commands cause a switch to the secondary input file, or back to the primary file, as appropriate. The prompt used by ECCE indicates which file is being operated on.

A subsequent `%S=filename` command, where the `filename` specified is different from the previous one, causes a change of secondary input file, and ECCE goes into (or remains in) secondary input mode.

Note that when ECCE is invoked the secondary file is defined by default to be the same as the primary file. Thus if in the first use of `%S` no filename were specified, the *original* version of the primary file (i.e. prior to the current edit session) would be used.

A secondary input file must:

* exist

* be a character file (or a character member of a group)

* be accessible to the ECCE user.

Otherwise a failure message will be output when the `%S=filename` command is issued.

A `%S` or `%S=filename` command must be the only command in the command line.

A `%C` command issued in secondary input mode has the same effect as in primary input mode, and ends the editing session in the same way.

While in secondary input mode ECCE maintains a second file pointer to the secondary input file, and editing commands cause this second file pointer to be moved. The main file pointer remains at the point it was at when the `%S` command was issued. If secondary input is re-entered then the secondary file pointer is where it was when secondary input mode was last left. Initially the secondary file pointer is at the beginning of the secondary input file.

Parts of the secondary input file can be transferred either to the main file or to a secondary output file (see Section 5.18). This is done by means of two commands: Note and Abstract, which can only be used in secondary input mode. The effect of Note is to note the current position of the (secondary) file pointer. This overrides any previously issued Note commands. The secondary file pointer can then be moved to the end of the section of the file which is required. The effect of an Abstract command is then to copy all text from just after the last Noted position to just before the current position of the (secondary) file pointer, either into the main file in a position just before the (main) file pointer, or to the end of the secondary output file if one is currently defined (see below).

The Abstract command may be repeated, in which case another piece of text is abstracted, starting at the same Noted position as the previous abstracted text; it follows that at least some of the text in the secondary file would be extracted twice by this operation.

Note that when an Abstract command is issued, the secondary file pointer must be *after* the last Noted position, otherwise it fails.

Only a subset of ECCE commands may be used in secondary input mode: in particular, commands which insert or delete text are prohibited. The simple commands allowed are F, L, M, M-, O, P, Q, R, T, V, W, X, Y, and Z (plus A and N of course). Monitoring and macro commands and programmed command qualifiers are also allowed. The H command can be used but will fail if it is necessary to insert spaces to move the pointer to the column specified.

Note that in general the size of the main file may not be increased by more than 262144 characters in a single editing session. This limit will be reduced if it would cause the maximum file size imposed by EMAS-3 on the user's process to be exceeded. It is possible to reach the limit using the secondary input facility.

5.18 SECONDARY OUTPUT

A command of the following form can be given:

```
%O=out
```

where "out" is a filename or output device. It is then defined to be the "secondary output file". If a file of the specified name already exists then its old contents are overwritten; otherwise it is created. It is *not* valid to specify a secondary output file as %O=file-MOD.

The effect of this command is that text selected from the secondary input file by the use of the N and A commands is copied to the secondary output file, not to the main file.

A subsequent %O=out2 command causes the current secondary output file to be closed and a new one to be defined.

The command %O (i.e. no secondary output file defined) causes the current secondary output file to be closed, and the main file to be reselected for receipt of text abstracted from the secondary input file.

Examples

```
>%O=.LP23                          /define secondary output
>%s=MISCIMPS                       /enter secondary input mode
>>f/READ TAG/                      /move to position
%ROUTINE  READ TAG(%INTEGER NAME)
>>l*                               /to get to start of line
>>n                                /note the position
>>(v/%END/\m)0m                    /move past end of routine
                                   /blank line following %END statement in file
>>a                                /abstract the routine
>>%s                               /and back to main source
>                                  /usual command prompt
```

The effect of this example would be to list the source of routine READ TAG on line printer 23. If the %O=.LP23 command at the start had been omitted, the source of the routine would instead have been copied into the main file, just before the main file pointer.

The dialogue of the above example might be continued as follows:

```
>%O                                /close secondary output file
>m-*                               /move to start of main file
%BEGIN                             /first line of file
>%S                                /switch to secondary input (file MISCIMPS)
                                   /current line (blank) in secondary input file
>>m-*nf'!END'                      /Note start of file, then seeks text '!END'
!END SPECS                         /text found
>>a                                /Abstract text. Note that the
                                   / file pointer is at the start of the line,
                                   / so the current line is not itself abstracted.
>>%s                               /switch back to the main file.
%BEGIN                             /current line in the main file
> ...                              /continue editing main file
```

5.19 ANCILLARY FUNCTIONS

Apart from its primary purpose of amending the contents of character files ECCE has four additional functions. These are:

* Facilities to let you format a paragraph of text with automatic line filling and optional right hand justification.

* A spelling checker.

* A syntax checker for use if the file you are editing is an IMP source file.

* A facility to allow you to call EMAS commands whilst still within ECCE.

5.19.1 Formatting your text

The simple command A and the special command %A can be used to ADJUST (format) a paragraph of your text. Note that A is used for ADJUST only when in primary input mode. It is used to ABSTRACT text when in secondary input mode.

The A command formats the current paragraph (lines delimited by blank lines or beginning or end of file), according to %A parameters (see below). No adjusting is done if the current line is blank. In either case A moves the file pointer to the start of the next non-blank line.

The %A special command is used to specify the way you want the A command to work. Parameters are specified in the form:

```
>%A=pstart, start, line, justify, print, sgapxs
```

E.g. `>%A=0, 0, 75, 1, 0, 1` (these are defaults)

Any parameters not specified retain their former values. Parameters cannot be set negative; any attempt to do so is ignored.

`pstart` Starting column position for first line of paragraph. (Column positions go from 1 upwards.) If `pstart` is set to 0 then the current start column of the *first* line of the paragraph is used.

`start` Starting column position for subsequent lines. If `start` is greater than `pstart`, so that the first line of the paragraph is 'outdented' with respect to the remainder, then no spaces are added or removed from the outdented part of the first line when the paragraph is Adjusted. If `start` is set to 0 then the current start column of the *second* line of the paragraph is used.

`line` Column position for end of line (maximum). If `line` is set to 0 then the current end column of the *first* line of the paragraph is used.

`justify` =0 lines filled but not right justified
#0 lines filled and right justified.

`print` =0 paragraph not printed as it is Adjusted
#0 paragraph printed as it is Adjusted.

`sgapxs` The number of *extra* spaces to be inserted between sentences.

Parameter settings are remembered by the Profile (%P) facility. Specifying `%A` without any parameter values causes the current settings to be printed.

5.19.2 The Spelling Checker

ECCE incorporates a spelling checker which can be accessed by using the simple commands Q and O. The spelling checker uses two dictionaries. There is a System dictionary which is maintained by the ERCC Documentation Officer. In addition each user can have their own dictionary which will contain words they have explicitly chosen to add to it, by using the ECCE command O.

The Q (Query spelling) command

The command Q is used to Query spelling. It moves the pointer through the file to the next questionable word (i.e. not found in the System dictionary or the user's private dictionary). A word is any sequence of characters delimited by spaces or newlines and discarding appended punctuation. Case is ignored as are one-letter words. Once a questionable word has been found it will not be found again in the same session. Q can be followed by S, the text replaced being the questionable word.

Following Q, F" can be used to find subsequent occurrences of the word.

Any digits following Q specify the range of the search in lines. Thus Q3 looks for a questionable word in the next three lines. It does not execute Q three times.

The O (Orthography) command

The O command can only be used following Q - thus repetition (e.g. O4) is not allowed. It causes the word found by Q to be put into the user's private dictionary - text file :ECCE#TDICT in the user's process. As this is a text file it can be examined and modified in the usual way.

If Q is not followed by O then the word is not put into the user's dictionary.

5.19.3 The IMP syntax checker

If you use the ECCE command to create and edit IMP source files you can use the built in syntax checker to speed up program development. The syntax checker is called by using the ECCE command %V. When the command is called the position of the file pointer is irrelevant. If any error is detected, the file pointer is moved to it and an error message is given. You can then correct the error and give the %V command again. This procedure can be repeated until %V detects no errors, in which case it leaves the file pointer at the end of the program.

> %V does not take account of the declaration of variables, and thus will not detect 'name not set', 'name set twice', etc.
>
> %V does check that %begin/%end, %start/%finish and %cycle/%repeat are properly nested. If a nesting error is detected, a summary of the current incomplete nesting structure within the current routine (or function or map or main program) is given.
>
> The structure of the program is saved at regular intervals, as the syntax checker proceeds, to help speed-up subsequent uses of %V (e.g. after a detected error has been corrected).
>
> %V can be applied to secondary input files.

5.19.4 Calling EMAS commands from within ECCE

Sometimes it is useful to use another EMAS command whilst you are editing a file. For example you may wish to check the name of another file for use for secondary input, or you may wish to respond to a TELL message. The method is to type an exclamation mark followed by the command and its parameters in response to the '>' prompt. For example:

```
>!FILES X*
```

It is even possible to call ECCE from within ECCE by using this mechanism. This can be used to edit a file to be used with the %I facility, for example.

5.20 The command SHOW

SHOW file

The SHOW command is used to call the editor for the purpose of examining a file rather than altering it. It takes a single parameter - the name of the file to be examined, with a default of T#LIST, the default compiler listing file (see Section 7.2.1). For example:

```
Command: SHOW TEST
```

A number of standard ECCE commands are precluded within SHOW, in particular those which attempt to alter the file. The following basic commands are allowed:

A, F, H, L, M, M-, N, O, P, Q, R, T, V, X, Y, Z

The monitoring and macro commands are allowed as well as programmed command qualifiers. As before, %C is used to close the session.

The `%O=out` and `%O` commands are allowed. In this case the text copied into the %O file is obtained by applying the N and A commands to the file being examined, *not* to a secondary input file. `%S` is *not allowed* in SHOW.

5.21 The command RECAP

RECAP

This command is used to interrogate the file containing a copy of all interactive terminal Input/Output operations for this user. The RECALL keyword of command OPTION must have been set to either PERM or TEMP (see Section 17.9) to specify that the terminal monitoring facility is required. RECAP is used without a parameter, and its operation is identical to that of SHOW.

5.22 SUMMARY OF ECCE COMMANDS

5.22.1 Simple commands

A	Adjust the current paragraph according to the %A parameters when in Primary Input mode. Abstract all text between the last Noted position and the file pointer when in Secondary Input mode.
B	Break. Insert a newline at the current position.
C	Cause Case inversion of the next character if a letter.
C-	Cause Case inversion of the previous character if a letter.
D/TEXT/	Delete the first occurrence of TEXT on the current line.
E	Erase the next character.
E-	Erase the previous character.
F/TEXT/	Find the first occurrence of TEXT.
G	Get a line of input.
Hn	Move Horizontally to before column n on the current line, inserting extra spaces if required.
I/TEXT/	Insert TEXT at the current position.
J	Join the current line to the next line.
K	Kill the current line.
L	Left shift the file pointer by one character position.
M	Move the file pointer to the next line.
M-	Move the file pointer to the previous line.
N	Note the current position.
O	OK the spelling of the word just queried by Q.
P	Print the current line.
Q	Query the spelling of the next unknown word in the file.
R	Right shift the file pointer by one character position.
S/STR/	Substitute STR for the TEXT just Found, Uncovered or Verified.
T/TEXT/	Traverse TEXT. Move the file pointer past the next occurrence of TEXT on the current line.
U/TEXT/	Uncover TEXT. Remove all characters between the file pointer and the next occurrence of TEXT on the current line.

V/TEXT/ Verify that TEXT occurs immediately to the right of the file pointer.

W Macro invocation.

X Macro invocation.

Y Macro invocation.

Z Macro invocation.

^ Mark the current position. Second use on the same file line defines a string which can be referred to by @.

5.22.2 Special commands

%A Assign parameters for ADJUST (A) command.

%B Write BACKUP copy to file ECCE#BACKUP.

%C Close the editing session.

%D Control the output of Data (non-printable) characters.

%F Full monitoring.

%H Get Help information.

%I Obey file of Interactive ECCE commands.

%L Lower case terminal.

%M Monitor normally (default).

%O Secondary Output mode.

%P Save Profile of currently defined macros and modes.

%Q Quit editor without writing output file.

%R Retrieve state of file before last file changing command line.

%S Secondary input mode.

%T Terse monitoring.

%U Upper case terminal (default).

%V Verify syntax of IMP program.

%W Macro definition.

%X Macro definition.

%Y Macro definition.

%Z Macro definition.

%^ Unwind editing to before last command line.

5.22.3 Programmed command qualifiers

( ) Bracket a group of commands for purposes of repetition and failure.

? Optional execution of command.

\ Invert the failure condition (succeed if and only if the command fails).

, Alternative execution (IF-THEN-ELSE...).

" Used in place of text string - meaning same text as used in last call of this command.

@ Refers to text bounded by use of ^....^. Used instead of text string e.g. `I@, F@, U@` etc.

&..& Delimit string of hexadecimal digits.

!command Invoke an EMAS command during editing; ECCE can be called recursively in this way.

CHAPTER 6
SOFTWARE PACKAGES on EMAS-3

6.1 Introduction

EMAS provides an ideal environment for developing your own programs, and the facilities provided to do this are described in this Guide. However, you may be able to avoid writing your own programs by using programs and packages which are already available on the system. This chapter discusses the sources of such programs and packages and tells you how to find out whether there is anything suitable for you.

6.2 Terminology

While there is no precise definition for words like 'package' in this context, here are some general guidelines:

* A program carries out a single task.

* A package, or software package, is a group of related programs. There will usually be a common theme - for example a statistical package will consist of a set of statistical programs. Further, the programs will be accessed by using a consistent format of control statements, and will expect their data to be set out in a consistent layout.

* A library, in this context, consists of a collection of procedures and functions which can be called from programs you write yourself. Sometimes the library procedures will carry out complex operations. In this case your program will consist of code to read in your data, call the procedure, and print the results. In other cases the library procedure or function will only perform a simple task, such as calculating a trigonometric function. In this case your programming will probably be more complicated.

6.3 Sources of programs, packages and libraries

Major packages and libraries are moved to EMAS by ERCC and CAST (Centre for Applications Software and Technology). Apart from these two sources, other items are imported by individual users or by departments who have a particular interest in making them available on EMAS. If you know of a package or library which you feel would be of widespread interest to EMAS users, then you should ask ERCC and CAST whether they would be willing to obtain it and mount it on EMAS. Whilst neither organization has unlimited resources for this type of work, they are both willing to consider requests from users.

6.4 Importing software yourself

If you cannot find someone else to import a package or library then you could consider doing the work yourself. However, you should understand that there may be substantial problems, for example:

* You will have to obtain the agreement of the owner or originator of the software to move it to EMAS. Software is a valuable commodity so licence fees can be very large.

* There may be serious technical problems involved in moving the software. This will depend upon the programming language used to write the software and the similarity between the operating system it is running on and EMAS (see Section 6.7 below).

* Documentation may be a problem. Even if you can obtain documentation for the software itself you will probably need to produce local documentation to describe the EMAS-3 implementation.

* You will have to decide how you will support the package after you have installed it. Who will investigate suspected faults? Who will mount new versions of the software as they become available? This support becomes increasingly important if other users come to rely on the software.

Before embarking on an exercise like this you should seek advice from the ERCC Advisory service. See also Section 6.6 below.

6.5 Finding out about available Software

As described in Section 6.3, there are many organizations and individuals involved in providing software on EMAS. There is no single source of information about all the software which is available. There are, however, a number of sources of information which you may find useful:

* User Notes - some of these cover single items of software - others provide a brief description of software available for a particular task or subject. Since the User Notes are being altered and added to very frequently it is not appropriate to include a detailed list in this Guide. You would be wise to start your search with User Note 0. This contains a list of all the current Notes, and an index to the material contained in them.

* Packages mounted by CAST have on-line help information. This is, at present, separate from the standard EMAS HELP mechanism. To find out which CAST packages are available you should type the following command:

```
Command: SEARCHDIR PLULIB:PACKDIR
```

  (This only needs to be done once. See Section 7.5.4 for further information about the SEARCHDIR command.)

  Then type:

```
Command: PACKHELP
```

  To obtain more information about a particular package type, for example

```
Command: PACKHELP SPSSX
```

* HELP (described in Section 2.7.4) is the main on-line documentation system for EMAS. You can try using HELP for a particular package or library name. Also you can use the name of a topic. e.g.

```
Command: HELP GRAPHICS
```

* The ERCC Advisory service may be able to help you to find a suitable program, package or library. Advisers have access to other staff in ERCC and CAST if they are unable to deal with your query on their own.

* The *Computing Newsletter*, published monthly by ERCC, includes articles about new items of software which have been mounted on EMAS. Each edition contains a cumulative index.

6.6 CONLIB - the Contributed Library

CONLIB is the name of a collection of software items which have been contributed by users. They range from individual procedures and simple commands up to complete packages. CONLIB is managed by ERCC in that it receives the items and makes them available to all users. It does not, however, provide full support itself. Rather, it expects the contributor to deal with queries from users, and to investigate and, if necessary, correct any faults.

6.7 Importing Load Modules from IBM systems

EMAS-3 runs on hardware which conforms to the IBM XA architecture. This type of computer is used very widely elsewhere with operating systems other than EMAS. One such operating system is IBM OS/MVS. There is a package on EMAS called VSS (Virtual Storage Simulator) which allows users to execute load modules imported from an IBM OS/MVS system. If you have access to such load modules for programs which would be useful to you, you should contact the Advisory service for further information.

6.8 Commercial use of software

Some software which has been imported to EMAS has limitations in its conditions of use whereby it can only be used for academic purposes. Commercial users should check on the legality of using imported software.

6.9 Criteria to use when selecting Software

Here are a few points you should consider when you are planning to use a package or library:

* Does the package do the job you need it for?

* Is there adequate documentation and support whilst you are learning to use the software?

* Who will investigate faults and give you support if you have problems whilst using the software? If your use is likely to continue for several years, will that support continue to be provided?

* Will you be able to exchange intermediate results with colleagues using the same software at other institutions?

* If you need to involve other staff, is there any training available or can they learn to use it from available documentation?

* Will you need to use the latest versions of the software as they become available? Will they be mounted on EMAS sufficiently quickly?

CHAPTER 7
COMPILERS, OBJECT FILES AND PROGRAM LOADING

7.1 INTRODUCTION

EMAS-3 includes compilers for the programming languages IMP and Fortran. The relevant language manuals contain full details of the languages. The environments provided by EMAS-3 for programs written in these languages are described in Chapters 8 and 9 respectively.

This chapter first describes the commands that invoke the compilers, and the associated commands PARM, INITPARM, LINK, MODIFY and RUN. There then follows a description of the loading of programs prior to their execution.

7.2 COMPILING PROGRAMS

7.2.1 The commands IMP and FORTRAN

```
IMP source, object, listing, errors

FORTRAN source, object, listing, errors
```

These commands are used to compile IMP and Fortran programs, respectively. In each case the four parameters are:

1. The name of the source file containing the program or routines to be compiled. Several files can be concatenated with the "+" operator; for example `DECLARS+B+C`.

2. The name of the object file to be generated by the compiler. Object files are described in Section 7.3. If no file of the given name exists one will be created. If the file already exists the action depends upon its type. If it is an OBJECT file then its contents will be overwritten by the compiler. If it is of any other type then the compiler will generate a failure message and stop. This is to reduce the risk of corrupting a valuable file by mis-typing its name. If this parameter is omitted, then a file named T#OBJECT is created.

   The keyword .NULL is a valid alternative if no object file is required. This facility reduces the compilation time, which is useful if the program is known to contain faults.

   Note that this parameter cannot specify a member of a partitioned file. An object file can, however, be copied into a partitioned file and used from there (see below).

   If the compilation succeeds then the resulting file is an OBJECT file. Otherwise there is no resulting file.

3. The name of a listing file or device. If this parameter is omitted a file with the name T#LIST is used. If a filename is used and the file already exists then it must be of type CHARACTER, otherwise the command will fail. Again, .NULL can be used if no listing is required. Note that T#LIST is destroyed when you log off, and is overwritten if used in another compilation in the current session.

4. A supplementary output file or device, for error messages only. By default this is defined to be .OUT, the interactive terminal or batch job journal. If a file is used and it already exists it must be of type CHARACTER, otherwise the command will fail.

Examples of calling compilers

```
Command: FORTRAN FORTP4,P4Y,.LP
```

This compiles the Fortran source file FORTP4 into an object file P4Y and outputs a listing on the local line printer.

```
Command: IMP A,AY
```

This compiles the IMP source file A into an object file AY and generates a listing file with the name T#LIST. Note that if T#LIST contains a listing from a previous compilation in the current session it is overwritten.

```
Command: IMP ERCC77:BASE+MINE,ABCOBJ,.LP14,FAULTS
```

This compiles the source file contained in files ERCC77:BASE and MINE into the object file ABCOBJ, producing a listing on the remote line printer .LP14 and a list of compile time faults in the file FAULTS.

7.2.2 The command PARM

```
PARM parmlist
```

Each EMAS-3 compiler offers a set of compile time options. A description of the options which can be used with each language appears in each of the chapters 8 and 9. Options are selected by means of a call of the command PARM. A call of PARM takes effect for all future compilations until you call PARM again or log off, whichever is sooner. PARM has the effect of resetting all values to the default settings, and then setting the ones selected. Hence

```
Command: PARM
```

with no parameters merely resets the defaults.

The parmlist consists of a list of keywords separated by commas. The meanings of these keywords vary slightly depending on which program language you are using. The details are given in the chapters relating to individual programming languages.

The form

```
Command: PARM =
```

can also be used; it causes the currently selected options differing from the default selection to be printed.

Notes

* The options which affect the compilation process must be specified *before* compilation.

* Although the existing options are unlikely to change, new ones may be introduced. Such changes will be advertised in the usual way; see Section 2.7.5 for details.

Example:

```
Command: PARM NOARRAY,NODIAG,DYNAMIC
```

7.2.3 The command INITPARM

```
INITPARM parmlist
```

This command is used to specify the PARM settings which you wish to use for most compilations. The effect of calling, for example:

```
Command: INITPARM NOARRAY,NOCHECK
```

is that an automatic call of:

```
Command: PARM NOARRAY,NOCHECK
```

is performed every time you log on. Having established your initial PARM setting in this way, you can change it for particular compilations by using PARM as described above.

The form

```
Command: INITPARM =
```

can be used to determine your current initial PARM setting.

7.3 OBJECT FILES

7.3.1 The structure of an OBJECT file

Object files are generated by the various language compilers in a standard format. One result of this is that it is possible, subject to the limitations of the parameter passing facilities of each language, for a procedure written in one language to call a procedure written in another. Thus an IMP program could call a Fortran subroutine.

An object file consists of several distinct areas. Most of these are not changed when the program is executed. In order to prevent *any* part of the object file being changed by execution, those areas which could change are copied at load time into a separate area in your virtual memory, and the copy is then used during execution of the program. The unchanging parts of the object file are accessed directly during execution.

Because unique copies of the unshareable parts of an object file are made at load time, it follows that if several users are executing the same program simultaneously (often true of utilities such as the editor), then they can all use the same copy of the object file. As a result there is a reduction in paging and in use of the main store. It is important to stress that the production of shareable object files is the compiler writer's responsibility and does not impose any restrictions on the user.

An object file can contain some or all of the areas listed below. The precise ways in which the compilers utilize these areas and the names they give to them vary, but one important distinction is made in every case: that between shared and unshared information.

* code — This is the main part of the file and contains the instructions which are to be executed by the computer. This area is shareable.

* constants — These are values which are used by the program but are not altered during its execution; e.g. IMP %CONSTANTINTEGERs. This area is shareable.

* initialisation values — These are values which are used to initialise variables which may be modified during program execution; the values specified in, for example, DATA statements in Fortran programs and %OWN variable declarations in IMP programs. This area is used only at load time, when the values are copied into an unshared area (not in the object file). For Fortran programs setting `LOADPARM ZERO` causes labelled COMMON areas not initialised by a BLOCKDATA to be filled with zeros by the loader; by default they are filled with the unassigned pattern. Note that blank COMMON and initialised labelled COMMON areas will be set to zero by the loader if `NOCHECK` was selected at compile-time.

* diagnostic tables — These are used to provide run-time diagnostic information in the event of a failure or an explicit call on the diagnostic package. This area is shareable.

* load data — This contains the names of entry points (procedure names) in this object file and the references made by this file to other object files. This information can be listed by using the command ANALYSE (Section 4.3). The load data is used by the program loader. It is shareable.

* history — This contains, in a packed format, information about the history of the contents of the file. It includes the programming language, the compiler version, and the time and date of compilation. This information can be printed by using the `H` option in the ANALYSE command (see Section 4.3).

7.3.2 Linking OBJECT files together

The command LINK is used to link (up to 256) object files to produce a compound object file. This may be useful in reducing the number of separate files in your file index. Note, however, that since program linking is automatic when a program is being run, it is usually unnecessary to use this command.

The command prompts for input files by using the prompt `Link:`. The list must be terminated with `.END`, whereupon the prompt `Object:` will be typed, to which the reply should be the name to be given to the single output file. The input filenames and the output filename must all be different. If the name chosen for the output file is that of an existing file its type must be OBJECT.

The following example shows how the command can be used:

```
Command: LINK
Link: P4Y
Link: ABCOBJ
Link: PDFILE_MEM1
Link: .END
Object: LINKFIL2
```

Alternatively, where it is more convenient to permanently store a series of LINK instructions, these can be placed in a file, and passed to LINK as its controlfile parameter:

```
Command: LINK CONFILE
```

A confirmatory message is normally printed. An optional second parameter, the keyword MAP, causes a short link map to be printed as well. This contains information about the relative start addresses of the object files in the combined file.

Note that it is not possible to extract a constituent object file from a linked file.

7.3.3 Altering OBJECT files

The command MODIFY can be used to alter the contents of an object file. It is used mainly to alter the loader data, the information used by the loader when loading the file. Amongst other functions, MODIFY can be used to:

* rename code and data entries and references

* mark some, or all references to be satisfied dynamically (see below)

* process the file to make it load faster (this is normally only appropriate for very large object files).

MODIFY takes two parameters. The first is the name of the OBJECT file to be modified. If a second parameter is provided then it is the name to be used for the modified file. By default the second parameter is the same as the first, i.e. the modified object file is written over the original.

Examples:

```
Command: MODIFY ERCC06:MYSSY,NEWMYSSY

Command: MODIFY LPUT17Y
```

MODIFY is described in detail in User Note 4. The following example demonstrates a common use of MODIFY to alter the names of two code entries to avoid a clash with standard entry names:

```
Command: MODIFY PRN85Y
Operation: RENAME
Proc ent pair: PLOT,MYPLOT
Proc ent pair: DRAW,MYDRAW
Proc ent pair: .END
Operation: CLOSE
OK
```

7.4 EXECUTING THE CODE IN AN OBJECT FILE

There are two ways to execute the code in an object file. The command RUN is used to execute the code in an object file which contains a complete program. An alternative method is to write a file which contains one or more external routines which can be called as commands. Chapter 18 explains how to write such commands.

7.4.1 The command RUN

```
RUN
```

The command RUN is used to load and execute a compiled program. It takes one parameter, the name of an object file which contains the compiled program. If this is omitted, the file T#OBJECT is executed (this is the default name of the file output by

the compilers). An object file member of a partitioned file can also be specified.

Examples:

```
Command: RUN MYPROG

Command: RUN EGNP99:SUMXOB_GO22
```

Before running a program it may be necessary to use the command DEFINE to establish links between I/O channels and particular files or devices (see Section 13.6). A call of CPULIMIT may also be required to set an appropriate time limit (see Section 17.7).

When you wish to run a program belonging to another user, the object file must be permitted to you in READ mode.

7.5 PROGRAM LOADING

An important part of EMAS is the "program loader". This is used to modify the contents of the virtual memory in preparation for the execution of a program or a command. The program loader is called automatically by the following:

* The part of the Subsystem which reads and interprets commands (Section 17.2) when you type a command, or a command is specified in the job stream, which is not already loaded.

* The command RUN, described above.

* The routine EMAS3 which is used to call commands from within programs (see Sections 8.5 and 9.4.5).

7.5.1 Loading a single object file

The process of loading a single object file involves the following actions:

* Connecting the file in the virtual memory (see Appendix 1). The address at which the file is connected is not important.

* Copying the initialisation values from the object file to your own work area.

* Establishing links, as necessary, to other procedures that are already loaded.

* Adding to the list of names of procedures still to be loaded.

* Relocating addresses in the program's work area, to reflect the addresses at which the object file and the work areas are connected.

7.5.2 Locating a procedure

A required procedure is located by one of two methods. When the command RUN is used the parameter for the command is the name of an object file or member which contains a compiled program. Thus the object file is explicitly named. In the case of a command, however, or a reference from another object file, the loader is only told the name of the required procedure, not the name of the object file containing it.

A DIRECTORY file contains a list of object files and the procedures contained in each. You can specify a list of directory files which are to be scanned during loading.

This is known as the 'Searchlist', and when the program loader is looking for a particular procedure, it examines each directory file in the list in turn until it finds it. It then loads the corresponding object file.

If the loader does not find a reference to the procedure within any of the directories in the Searchlist, an error message is produced and the loading process is abandoned. See LOADPARM (below) for a way to get around this.

7.5.3 The directory search order

The following directories are searched to locate a procedure:

* The session directory: this is a directory containing information about all the procedures which are currently loaded. At the start of the session this comprises the procedures in the Subsystem "basefile" only; these include most of the commonly used commands. Later during the session, as other Subsystem components are used (such as the compilers or the mathematical routines), they are added to the session directory.

* Next, your active directory is searched (see below).

* Next, each directory in the Searchlist. These are directories which have been specified by the SEARCHDIR command (see below).

* Finally, a directory of Subsystem procedures is searched. This contains references to object files containing all the Subsystem procedures not in the basefile.

There are a number of implications of this search order:

* Since the session directory always contains references to the procedures in the Subsystem basefile, these cannot be replaced by references to procedures of the same name belonging to you.

* Subsystem procedures other than those in the basefile can be overridden *only* if they have not already been loaded during this session.

* The number of directories to be searched affects the speed of the loader. It is advisable to check the Searchlist periodically, by using

```
Command: SEARCHDIR =
```

  and to remove any redundant entries. In addition, the list should be ordered so that the most frequently used procedure names are in the active directory or the SEARCHDIR directory nearest the top of the list.

7.5.4 The command SEARCHDIR

This command is used to add a directory to the Searchlist (see above). It takes the name of one directory, and optionally a number in the range 1-16 to specify the position it should occupy in the list. The default is 1, i.e. the top of the list. Alternatively SEARCHDIR can be used with the parameter '=' to print the current Searchlist.

Examples:

```
Command: SEARCHDIR CONLIB:GENERAL
```

```
Command: SEARCHDIR ERCLIB:NAGLIB,3

Command: SEARCHDIR =
```

Note that if an attempt is made to add a directory to the list which is already in the list, then it will be relocated at the position specified.

7.5.5 The command LOOKFOR

One of the problems with using the command SEARCHDIR is knowing which directories contain the procedure entry names or command names you require. Normally you will be able to obtain this sort of information from the printed or on-line documentation which describes the facility you wish to use. An alternative method is to use the command LOOKFOR. This is only of use if you know the name of the entry or command you require. The command takes three parameters:

```
LOOKFOR entry, type, dirlist
```

`entry` - is the name of the command or procedure you wish to find.

`type` - can be `'PROC'` or `'COMM'` (either of which can be abbreviated to a single letter) to indicate whether you are looking for a procedure or a command. If this parameter is omitted then LOOKFOR will assume you are trying to locate a command.

`dirlist` -normally this is omitted since the default action is to search a list of all the commonly used public directories. You can create your own `dirlist` file which should contain a list of directory filenames, one to a line. Specify the name of this file to cause LOOKFOR to interrogate those directories. In all cases LOOKFOR will look at the list of currently loaded entries and at entries in the current ACTIVEDIR.

LOOKFOR will list the names of directories it is searching and will indicate which, if any, contain the entry name you need. Note that command entry names always begin with `"C#"`, so, for example the command METER appears in the output from LOOKFOR as `"C#METER"`.

7.5.6 The command REMOVEDIR

This command is used to remove one or more directories from the Searchlist (the list of directories searched by the loader for commands and procedures). If you wish to remove more than one directory you should separate their names with '&'.

Examples:

```
Command: REMOVEDIR CONLIB:GENERAL

Command: REMOVEDIR ERCLIB:GRAPHICS & ERCLIB:NAGLIB
```

7.5.7 The command LOADPARM

The command LOADPARM allows you to control the behaviour of the program loader. It takes one or more of the following keyword parameters, separated by commas. As with the command PARM, this command remains in effect until it is called again or until you log off.

LET - Normally the loader will not allow a program or command to be executed if

there are any unsatisfied references. If LET is specified then the program will be allowed to run. It may subsequently fail with an error message if an unsatisfied reference is actually called.

MIN - Normally the loader completes the loading process by satisfying all references from the initial file (the one being RUN, or containing the selected command) and, if this involves loading further files then all the references from them are satisfied too. This process continues until all the files required have been loaded. If MIN is specified then the loader will only load the specified file and create any common areas it requires. Any code references are changed to dynamic and will be satisfied during execution only when actually called.

ZERO - Normally un-initialised COMMON areas are filled with the unassigned pattern. If ZERO is selected then they will be filled with zeros.

Example:

```
Command: LOADPARM LET,ZERO
```

The form

```
Command: LOADPARM =
```

can be used to determine the current settings.

7.6 DIRECTORY FILES

A directory file contains two types of information:

* mappings between procedure entry names and the names of object files (and members) which contain the procedures.

* mappings between command alias names and the command names to which they refer (see command ALIAS, Section 17.6).

7.6.1 Creating directory files

Directory files are normally created automatically when they are required, but they can be created explicitly by use of the command NEWDIRECTORY. The first parameter is the name to be given to the new directory. It must not be the name of an existing file. The second parameter is optional, and determines the size of the directory file in pages (units of 4096 bytes). The default is 1 page and the maximum allowed is 32 pages.

Example:

```
Command: NEWDIRECTORY MYDIR
```

7.6.2 The active directory and the command ACTIVEDIR

The current active directory is the directory which is modified by explicit or automatic calls of the commands INSERT, REMOVE and ALIAS (see below). It is also the first directory to be searched for the names of required procedures which are not already loaded. If one of the commands INSERT, REMOVE or ALIAS is called and the active directory does not exist then one is created automatically.

At log-on or at the beginning of a batch job the name of the active directory is :SS#DIR. The name of the active directory can be changed by use of the command ACTIVEDIR. It takes one parameter - the name of the directory to be used as the active directory for the rest of the session, or until ACTIVEDIR is used again:

```
Command: ACTIVEDIR MYDIR
```

7.6.3 The command INSERT

Use this command to put information about an object file into a directory. The information consists of the names of procedure and data entries contained in the object file. This information is used by the loader to locate a particular object file in order to satisfy a procedure or data reference from another file or from the command interpreter. It is only necessary to use this command for object files which are going to be referenced by their entry names. It is not necessary to use it for main program files which are RUN. Once a file has been "inserted", the details of its entries are automatically updated in the Active Directory each time it is re-compiled.

```
INSERT objectfile, directory
```

The command takes two parameters:

`objectfile` this can be a single object file, or a list of object files separated by & characters, or a wild-card expression (see Section 17.2)

`directory` is optional. It specifies the name of a directory into which the details of the object files are to be inserted. If it is omitted the information is inserted into the current ACTIVE directory.

Examples:

```
Command: INSERT ROUTY&ERCC18:FILEY,TESTDIR

Command: INSERT AY
```

After the explicit call of INSERT, the use of any of the standard compilers has the effect of re-INSERTing an object file after *successful* recompilation, if the new object file contains any new or changed procedure names. Thus if you insert the object file OBJECT23 into your current active directory once, and subsequently recompile it several times, the information relating to OBJECT23 will be kept up to date in the active directory.

If INSERT is used for an object file which is already inserted, the effect is to REMOVE the existing information and then to re-INSERT it, thus ensuring that the information reflects the current state of the object file.

7.6.4 The command REMOVE

Use this command to remove all the information about an object file from a directory. This reverses the effect of a previous call of INSERT. Consequently, the names of the procedure and data entries contained in the object file are removed from the directory.

```
REMOVE objectfile, directory
```

The command takes two parameters:

`objectfile` this can be a single object file, or a list of object files separated by & characters, or a wild-card expression (see Section 17.2)

`directory` is optional. It specifies the name of a directory from which the details of the object files are to be removed. If it is omitted the information is removed from the current ACTIVE directory.

Apart from explicit calls, REMOVE is invoked automatically when an object file is destroyed.

7.6.5 The command TIDYDIR

TIDYDIR is used to tidy a directory file. It takes one optional parameter - the name of the directory to be tidied. If the parameter is omitted then the current active directory will be tidied. This command is not required regularly. You should use it if you have any problems with loading programs, to ensure that they are not due to directory inconsistencies.

The process of tidying includes REMOVing all the inserted files and INSERTing them again. This ensures that the directory is up to date in two respects:

* Information about any object files which are no longer available is removed from the directory. This includes object files which have been deleted by the system after a period of non-use (see Section 3.8.1).

* Any inconsistencies between the list of entries for a particular file held in the directory, and the list actually in the file can be corrected. This can occur if, for example, a file is recompiled when this directory is not the current ACTIVEDIR.

7.7 DYNAMIC LOADING

Normally all the external references made by a program must be resolved (by the loading of appropriate object files) before the program starts execution. This can mean that large programs or packages take a relatively long time to start execution, because of the complexity of the loading process. With such programs, a number of entry points might never be called during a specific run, so that the time taken to locate and load the relevant object files has effectively been wasted.

It is possible, however, to arrange that a procedure (i.e. the object file containing it) is loaded only when it is called, rather than loaded automatically before execution of the program. This is known as *dynamic loading*, since it takes place when the program is executing.

It is important to appreciate that for a procedure to be loaded dynamically, it is not necessary to modify it in any way. It is the program or procedure that references the procedure that must be marked to inform the program loader that the reference is to be satisfied dynamically. This can be done at compile time, thus:

Fortran: compile with `PARM DYNAMIC` - *all* external references are then marked for satisfying dynamically.

IMP: use %DYNAMICROUTINESPEC in place of %EXTERNALROUTINESPEC (see Chapter 8) for all routines and procedures which are to be loaded dynamically. It is also possible to use `PARM DYNAMIC`, causing all external references to be marked for satisfying dynamically.

Alternatively, the MODIFY command (see above) can be used to mark one or more

external references in an object file for satisfying dynamically.

Another way to cause a program or command to satisfy its references dynamically is to use the command LOADPARM (see above) before calling it:

```
Command: LOADPARM MIN
```

To obtain information about the references in an object file, use

```
Command: ANALYSE objectfile, R
```

7.7.1 Running a program with dynamic references

A program or command containing dynamic refrences is called in exactly the same way as one containing only static references. When the file containing the dynamic references is loaded, any references to procedures already loaded are satisfied immediately. All others are left unsatisfied. The program then starts executing until a call is made on a procedure which is not loaded. At this point a jump is made to the loader and an attempt made to load the required procedure. If this fails then a message is printed of the form

```
Failed to load NAME dynamically.  Entry NAME not found.
```

where NAME is the name of the procedure in question. This is followed by diagnostics, and program termination. If the procedure is successfully loaded, it is then called. Subsequent calls behave exactly as for calls to procedures loaded prior to execution.

CHAPTER 8
IMP ON EMAS-3

8.1 INTRODUCTION

Most of the EMAS operating system and the Subsystem is written in the high level programming language IMP. Consequently EMAS provides an ideal environment for running programs written in this language. The language is described in the IMP80 Language manual. The ways in which IMP80 on EMAS-3 differs from earlier versions are described in User Note 58. This Note also explains how the utility program SOAP (see Section 8.9.1) can be used to automate much of the work of converting IMP programs from EMAS 2900 to EMAS-3.

8.2 COMPILATION

An IMP source file can be compiled by using the command IMP, as described in Section 7.2.1. The related command PARM, used to set compile time options, is described in Section 7.2.2.

PARM option	Notes
NOCHECK	By default a check is made that each variable whose value is about to be used has been assigned a value: this is inhibited by NOCHECK.
NOARRAY	By default an array bound check is carried out on all array accesses: this is inhibited by NOARRAY. Note that some checking on multi-dimensional array bounds will be carried out even if NOARRAY is selected.
NODIAG	By default symbol tables are retained at run-time, for the production of values in local variables in diagnostics: this is inhibited by NODIAG.
NOLINE	By default code is included to keep a record for diagnostics of the number of the line being executed: NOLINE inhibits this.
OPT	The compiler omits run time checks implied by the other PARM settings and produces optimised code for programs which have been thoroughly tested.
NOTRACE	By default routine traceback information for diagnostics is kept: this is inhibited by NOTRACE.
NOLIST	By default a program listing is generated at compile-time. NOLIST causes output of a heading and any error messages only.
DYNAMIC	This marks all references for satisfying dynamically. In addition, externalroutines are loaded dynamically if they are specified in the calling program as, e.g., %dynamicroutinespec. See also "Dynamic loading", in Section 7.7. PARM NOLINE is forced off.
CODE	This causes a listing of the object code generated by the compiler to be produced.
MAXDICT	This causes the compiler tables to be set to the maximum possible size.

Table 8.1 IMP PARM Options

8.2.1 The structure of an IMP source file

The IMP80 Language manual explains how you can write software in IMP as complete programs, or as files of external procedures. A complete program, once compiled, can be executed by use of the RUN command - see below. A file of external procedures can be used:

* to contain one or more routines and functions which are needed by more than one program.

* to contain procedures which are to be accessed as EMAS commands. Chapter 18 contains information about writing your own commands.

In order to access the external procedures in an object file, either as commands, or in order to call them from a program, the object file containing the external procedures must be INSERTed in a directory. This directory must be either the current active directory or a directory in the SEARCHDIR list. This is explained more fully in Section 7.5.

Note that the only information that is used to identify an external entry is its name. The compiler is able to check that procedure calls within a file are consistent with the %specs provided, but it is your responsibility to check that the %spec of an external procedure is consistent with that of the procedure being called.

8.2.2 External data

The IMP80 Language manual also contains information about sharing data between object files. As with external procedures it is important that you ensure that the %spec in one file is consistent with the declaration of the external data in the other file.

8.2.3 Calling Fortran

The object files produced by the EMAS IMP and Fortran compilers are compatible, and subject to certain limitations imposed by the languages it is possible to make cross calls from one language to the other. This is described in the Edinburgh Fortran77 Manual.

8.3 EXECUTING IMP PROGRAMS

The command RUN is used to cause the execution of an IMP program (see Section 7.4.1). Before starting execution the program has to be loaded, as described in Section 7.5.

8.3.1 Unsatisfied References

If, during loading, a procedure cannot be located to match an %external specification then an appropriate message is output. Execution of the program is then inhibited unless the LOADPARM option LET has been specified (see Section 7.5.7). In that case the program is allowed to continue until a call is made on the procedure. If this occurs an appropriate message is output, followed by an execution of %monitor; %stop.

If an %external data entry is not found to satisfy an %external data reference then a failure message is printed and execution is inhibited.

8.4 LIBRARY ROUTINES

8.4.1 IMP System Library

Standard IMP library routines are available in the System Library, whose directory is searched automatically following a search through any directory files nominated by the user (see Section 7.5.3). The library routines available are described in the IMP80 Language manual.

8.4.2 Graphics and other libraries

There are libraries of routines for using graphics devices and for other specialist purposes. In order to access such routines it is necessary to use the SEARCHDIR command, as explained in Section 7.5.4. There are suggestions in Chapter 6 about selecting such software.

8.5 ACCESSING EMAS COMMANDS FROM IMP PROGRAMS

It is possible to call EMAS commands from within a program. These can be commands which are part of the Subsystem or others written by yourself or another user. To do so you need to include a specification for the routine 'EMAS3'. The spec is:

```
%externalroutinespec EMAS3(%stringname COMMAND,PARAMS,
      %integername FLAG)
```

The first two parameters to this routine should be the name of the command and the complete parameter string that is to be passed to it. The third parameter is used to return the result of the command. The standard convention is that a value of zero indicates that the command executed correctly.

Examples of calls:

```
EMAS3("DEFINE","1,.LP15",f)

EMAS3("RENAME","TEMPAC,ACCOUNTS",FLAG)
```

8.6 ACCESSING OTHER EMAS FACILITIES FROM IMP

There are many Subsystem procedures which can be called by IMP programs. Some of the commonly used ones are described here. Others are described in Chapter 18 and User Note 80.

8.6.1 Use of name type parameters

A design feature of EMAS-3 is that it should be possible to call all interface procedures from several programming languages. This fact accounts for the wide use of name type parameters in the interface routines. These are used in cases where the program passes information to the procedure, as well as in the more usual case where the parameter is used by the procedure to return a result. This is done to make it possible to call the procedures from languages such as Fortran which pass parameters by name.

Whilst it is essential that you type the %external %specs correctly there is a relaxation of the rules for the checking of actual parameters against %specs. The rule is that, in the case of externally specified procedures it is permissible to use constants or expressions as actual parameters, even though the %spec contains name type parameters.

For example the spec of EMAS3PROMPT is:

```
%externalroutinespec EMAS3PROMPT(%stringname TEXT)
```

but it can be called, for example:

```
EMAS3PROMPT("Text:")
```

8.6.2 Setting the prompt text

EMAS3PROMPT is used to set the text which is printed on the interactive terminal for subsequent input requests. It must be explicitly specified:

```
%externalroutinespec EMAS3PROMPT(%stringname TEXT)
```

When you run a program the default prompt is `Data:`. A call of EMAS3PROMPT within the program will change this and the new text will be used as the prompt whenever input from the interactive terminal is required, until another call of EMAS3PROMPT or a return to command level is made. No text is printed as a result of the call of EMAS3PROMPT - it is the subsequent requests for input that result in the text being typed. As explained in Section 2.3.6, under some circumstances prompts will be printed after the required input has been typed. If this is a serious inconvenience to you, you should avoid typing ahead, and await the prompt for each item of input.

The following demonstrates the use of EMAS3PROMPT:

```
%begin
%externalroutinespec EMAS3PROMPT(%stringname TEXT)
%string(18) INFILE, OUTFILE
%integer RECORD
   EMAS3PROMPT("Input File:")
   READSTRING(INFILE)
   EMAS3PROMPT("Output File:")
   READSTRING(OUTFILE)
   EMAS3PROMPT("Records:")
   READ(RECORD)
   :
   :
```

Assuming that the program was compiled into an object file named `TRANSFORM`, when it is run from an interactive terminal the dialogue might have the following appearance:

```
Command:RUN TRANSFORM
Input File:"RES506"
Output File:"OUTPUT6"
Records:256
   :
   :
```

Notes

* The maximum length of a prompt is 63 characters.

* Any characters can be included in a prompt, including space, newline and even control characters such as BEL and CR.

* If a null string is specified no prompt is output.

8.6.3 Detecting multi-character Interrupts from the Interactive Terminal

The routine EMAS3INTERRUPT is used to enable a program to respond to interrupts typed at the interactive terminal. As explained in Section 14.6, normal processing can be interrupted by pressing a particular sequence of keys to generate the prompt `Int:`, to which the reply can be:

* <RETURN> - this causes the interrupt to be ignored.

* A single letter followed by <RETURN> - these single character interrupts are interpreted by the Subsystem (see Section 14.6.3).

* A reply of 2-15 characters, terminated by <RETURN> - the occurrence of such an interrupt can be detected by the currently executing user program, via EMAS3INTERRUPT.

EMAS3INTERRUPT must be explicitly specified:

```
%externalroutinespec EMAS3INTERRUPT(%stringname INTERRUPT)
```

It returns and *clears* the last multi-character interrupt from the interactive terminal. Note that there is no queue of such interrupts; thus if a multi-character interrupt has not been read (via EMAS3INTERRUPT) by the time that another multi-character interrupt is sent then it will be lost. EMAS3INTERRUPT returns a null string if there is no interrupt outstanding.

8.6.4 Obtaining information about the Subsystem

There is sometimes a requirement for a program to determine information about the environment in which it is running.

The externalroutines EMAS3UINFI and EMAS3UINFS can be used to obtain such information. The functions must be specified:

```
%externalroutinespec EMAS3UINFI(%integername ENTRY,VALUE)

%externalroutinespec EMAS3UINFS(%integername ENTRY,
      %stringname VALUE)
```

The result obtained by calling either of these procedures is a single item of information, specified by the value of `ENTRY`. The values of `ENTRY` which can be specified with these functions, and the meanings of the results obtained, are given in Table A5.1

Example

In the following program the user's name and delivery information are printed out, along with the number of user processes currently active on the System.

```
%externalroutinespec EMAS3UINFI(%integername ENTRY,VALUE)
%externalroutinespec EMAS3UINFS(%integername ENTRY,
      %stringname VALUE)
%string(31) HOLD
%integer U
      :
      :
   EMAS3UINFS(1,HOLD)
   PRINTSTRING("User: ".HOLD); NEWLINE
   EMAS3UINFS(2,HOLD)
   PRINTSTRING("Delivery: ".HOLD); NEWLINE
   EMAS3UINFI(3,U)
   PRINTSTRING("Current no of users:"); WRITE(U,1)
   NEWLINE
      :
      :
```

8.6.5 Checking for the existence of a file

It is often necessary to know whether a file of a particular name exists. The external routine EMAS3EXIST can be used. It must be explicitly specified:

```
%externalroutinespec EMAS3EXIST(%stringname FILE, %integername FLAG)
```

The first parameter must be a string or string expression containing the name of the file. The form "file_member" is also permitted (see Chapter 20). Upon return FLAG is non-zero if the file exists and is permitted to this user in at least one mode.

Example:

```
%integer RES
%string(6) USER
%string(255) FILE
%externalroutinespec EMAS3EXIST(%stringname FILE,
      %integername FLAG)
   READSTRING(USER)
   READSTRING(FILE)
   EMAS3EXIST(USER.":".FILE,RES)
   %if RES#0 %start
      PRINTSTRING(FILE." exists")
      NEWLINE
   %finish
```

8.7 IMP INPUT/OUTPUT

Four file access methods are provided for IMP:

* Streams, used for character input/output: use routines READCH, WRITE, etc.

* Sequential binary file handling: use routines READSQ, WRITESQ, etc.

* Direct access binary file handling: use routines READDA, WRITEDA, etc.

* Store mapping: see Chapter 19.

The routines and functions provided by the IMP language implementation are described in the IMP80 Language manual. The System-dependent aspects of these facilities are described below.

8.7.1 Linking channel numbers to files and devices

The command DEFINE (Section 13.6) is used to establish a link between a channel number used in a program and a particular file or output device. Note that the channel number must be a positive integer, normally not greater than 1000.

As explained in Chapter 13, the access method to a file or device depends on the way in which the associated channel number is used in the program. In the case of IMP programs, the access method depends upon which I/O routines are used with the channel number defined. Thus any conflict between a file and its access method (for example an attempt to use a character file, rather than a data file, for direct access) will only be faulted when the program is executed, not when the DEFINE command is given.

8.7.2 Character I/O

Character I/O can be used both for input and output.

For input the following can be used:

* The interactive terminal (defined if necessary as ".IN").

* A character file (see Section 13.3).

* A data file (see Section 13.4)

For output the following can be used:

* The interactive terminal (defined if necessary as ".OUT").

* A character file. If the specified file does not exist it will be created automatically. If the file does already exist it will be overwritten if it is already a character file. Otherwise the program will fail at the relevant call of SELECTOUTPUT, with an appropriate failure message.

* Any of the following output devices, or their remote equivalents:

  line printer .LP
  document printer .DP

  For further information about output devices see Section 15.6.2.

Examples of valid DEFINE calls for IMP Streams:

```
Command: DEFINE 1,.IN
Command: DEFINE 22,.OUT
Command: DEFINE 17,ERCC88:RESULTS.FILE2308
Command: DEFINE 80,.LP15
```

Default stream definitions

Table 8.2 shows the definitions that are established by default, in foreground and background mode:

Channel	Input/Output	Device	
		Foreground	Background
0	Input	.IN (user's terminal)	.IN (Job file)
0	Output	.OUT (user's terminal)	.OUT (Journal)

Table 8.2 IMP Default Channel Definitions

As indicated in Table 8.2, .IN, the primary input device, stands for the interactive terminal in foreground mode, and for the job file in background mode; .OUT, the primary output device, stands for the interactive terminal in foreground mode, and for the output destination specified when the job was initiated, in background mode (see Section 16.3).

Line length limits are not imposed by EMAS, but are in some cases by the devices themselves (for example line printers), or by the PAD (in the case of the interactive terminal).

Note that stream 0 is exceptional in that it can be used for input and output simultaneously.

8.7.3 Sequential Binary Files (SQFILES)

The routines OPENSQ, CLOSESQ, READSQ, WRITESQ and READLSQ are available to access sequential binary files. The only type of file that can be used for sequential binary input is a DATA file (see Section 13.4).

For binary output the file type must be DATA. If an existing file is defined and it is not a DATA file the program will fail when it attempts to open the file.

It is also possible to use magnetic tape for binary input and output (see Section 21.4).

8.7.4 Direct Access Files (DAFILES)

The routines OPENDA, CLOSEDA, WRITEDA and READDA are available for accessing direct access binary files. They can only be used in conjunction with data files having a fixed record length of 1024 bytes. When OPENDA is used on a channel for which a new file has been DEFINEd, a file is created and each byte of it is filled with a bit pattern which can be interpreted as meaning "unassigned". The size of the file is extracted from the SIZE parameter passed to DEFINE. For example, if it were required to create a file of 60 records (each of 1024 bytes), the following DEFINE could be used:

```
Command: DEFINE 70,TESTDA,60
```

Note that this command does not create the file `TESTDA` - it is also necessary to run a program that includes the statement

```
OPENDA(70);  ! Create file (if new) and open it
```

See also the section on the command DEFINE, in Section 13.6.

8.7.5 Store Map Files

Accessing files by mapping them onto variables and arrays in a program is described in Chapter 19.

8.8 EFFICIENCY OF IMP WHEN USED WITH EMAS

When writing IMP programs specifically for EMAS there are a number of ways in which greater efficiency can be achieved. The main aim should be to reduce the number of page turns. This is particularly important for programs run in foreground mode, since the number of page turns is an important factor in determining the elapsed time taken to run the program. The following points are suggested:

* Use a %constant rather than an %own array where the contents of an array remain constant throughout a program.

* Do not use %own arrays just to achieve initialisation to zero. Instead use a normal array and a cycle to clear it to zero.

* Use arrays of the correct size. If the size varies significantly from one run to another, use dynamic bounds.

* When accessing two-dimensional arrays remember that they are laid out in store in such a way that their first bound increases more rapidly; for example

```
%integerarray IN(1:200,1:100)
```

is laid out thus:

```
IN(1,1)
IN(2,1)
IN(3,1)
:
:
IN(200,1)
IN(1,2)
IN(2,2)
etc.
```

Where possible, when accessing a large array of this type systematically, an attempt should be made to access the elements in the order in which they occur in store. For example, when such an array is cleared to zero the following should be used:

```
%integer I,J
%for I=1,1,100 %cycle
      IN(J,I)=0 %for J=1,1,200
%repeat
```

These points should be seen in perspective. Most of them are only of importance in programs which access large data areas.

8.9 IMP PROGRAMMING AIDS

The following tools may assist you in developing IMP programs on EMAS-3.

8.9.1 The SOAP Formatting Program

This program can be used to format your IMP80 source files, to improve their legibility. It incorporates many user definable options which allow you to produce a format which suits your needs. SOAP also includes a facility to automate most of the work of converting IMP80 programs which have been written for EMAS 2900 to run on EMAS-3. SOAP is described in User Note 14. The conversion facility is described in User Note 58.

8.9.2 IMP80 Syntax Checker

The ECCE text editor, which is described in Chapter 5 of this Guide incorporates an IMP80 Syntax checker. If you use ECCE for editing IMP programs you will find this useful tool can speed up program development: see Section 5.19.3.

CHAPTER 9
FORTRAN ON EMAS-3

9.1 INTRODUCTION

This Chapter describes the environment in which a Fortran program runs on EMAS-3. A compiler is provided for the Fortran77 language, which is fully described in the Edinburgh Fortran77 Language Manual. User Note 57 provides information about moving programs to Fortran77 from earlier dialects of Fortran. Unlike the Subsystem interface on earlier versions of EMAS, that for EMAS-3 has been written in such a way that most of its routines and functions can be called from Fortran programs.

9.2 COMPILATION

The command Fortran is used to compile a Fortran source file, as described in Section 7.2. The command PARM is used to set compile-time options: it is described in general in Section 7.2.2. Table 9.1 contains a list of PARM options for Fortran.

The source file may contain one or more program units (subprograms), of which at most one can be a main program. If after successful compilation the object file is analysed by using ANALYSE, it will be found to contain the following:

* a MAIN PROGRAM entry, if the file contained a main program
* an entry for each SUBROUTINE, FUNCTION or ENTRY statement
* a data entry for each named COMMON statement in a BLOCK DATA subprogram.

PARM option	Notes
NOARRAY	By default Fortran checks that each subscript in an array reference is within the declared range for that dimension. NOARRAY inhibits these checks.
NOCHECK	By default a check is made that each variable whose value is about to be used has been assigned a value: this is inhibited by NOCHECK.
OPT	The compiler omits run time checks implied by other PARM settings and produces optimised code for programs which have been thoroughly tested.
OPT1	Implies OPT and performs partial source level optimisation within each subprogram.
OPT2	Implies OPT1. Also a high level of optimisation is performed within each subprogram.
OPTEXT	If OPT1 or OPT2 has been specified OPTEXT will produce a pseudo-Fortran listing of the optimised program.
NODIAG	By default symbol tables are retained at run-time for diagnostics: NODIAG inhibits the retention of these tables.
NOLIST	By default a program listing is generated at compile-time. NOLIST causes only erroneous statements to be listed.
NOCOMMENTS	Suppresses the output of comment messages.

NOWARNINGS	By default warning messages are generated at compile-time when any features not in the ANSI Fortran77 standard are used. NOWARNINGS suppresses the output of such messages.
STANDARD	If STANDARD is specified then an error is reported when any feature not in the ANSI Fortran77 standard is used.
R8	By default Fortran allocates 4 bytes to REAL variables. R8 provides greater precision by using 8 bytes for REALs, 16 bytes for DOUBLE PRECISION, and 8 bytes for each part of a COMPLEX variable.
I8	By default Fortran allocates 4 bytes to INTEGER variables. I8 allocates 8 bytes for each default INTEGER location to ensure the validity of EQUIVALENCE statements when R8 is used.
L8	By default Fortran allocates 4 bytes to LOGICAL variables. L8 allocates 8 bytes for each default LOGICAL location to ensure the validity of EQUIVALENCE statements when R8 or I8 is used.
DYNAMIC	DYNAMIC causes all references to be marked for satisfying dynamically. See also "Dynamic loading", in Section 7.7.
ATTR	At the end of each program unit, ATTR causes a list of attributes (type, size, whether scalar or array, etc.) to be output.
CODE	CODE causes a listing of the object code generated by the compiler to be produced.
XREF	XREF causes output of references to variables, with line numbers of their occurrences.

Table 9.1 Fortran PARM Options

9.3 PROGRAM LOADING AND EXECUTION

9.3.1 Subroutine linking

CALL statements and references to functions result in cross references being included in the load data part of the object file. It is important to appreciate that the only information included in the reference is the name - strictly, the first 31 characters of the name. There is no information held to indicate what type of subprogram is being referenced. Thus great care should be taken to specify the correct type and parameter list in subroutine and function calls. Calling the wrong type of subprogram, or using the wrong number or types of parameters, can result in faults that are very difficult to diagnose.

9.3.2 Data linking

The COMMON statement is used to provide access to data used in more than one subprogram. COMMON blocks in different subprograms are automatically equivalenced when a program is loaded. Each named COMMON block is equated with any other COMMON block of the same name, and is initialised if a BLOCK DATA subprogram is included which refers to the same named COMMON block. All named COMMON blocks with the same name must be of the same length.

9.3.3 Running Fortran programs

The RUN command is used to cause execution of a Fortran program. The parameter must be the name of an object file which includes a main program. Before starting execution the object file must be loaded, as described in Section 7.5. Normally all references to subroutines and functions are satisfied before starting execution, appropriate messages being printed for any that are not found (but see 'Dynamic Loading' in Section 7.7). If some are not found execution is only allowed to proceed if the `LOADPARM LET` has been specified (see Section 7.5.7). If it has been, processing will continue up to the first call of a routine which was not found. At this point an appropriate message is printed, followed by diagnostics and execution of a STOP.

It is also possible to execute a command written by a user in Fortran (see Chapter 18).

9.3.4 System library

A Fortran system library is automatically searched for the intrinsic and mathematical function subprograms. A table of these subprograms, showing what parameters they require, can be found in the Fortran77 Language Manual.

9.3.5 Mathematical, Graphics and other libraries

There are several libraries of routines available to the Fortran programmer on EMAS-3. For example, the Numerical Algorithms Group (NAG) Library provides a large number of facilities and is widely used within universities and Research establishments. There are several libraries of Graphics routines, some of which are described in the ERCC Graphics Manual. Further information can be found in the sources described in Section 2.7.1.

9.4 ACCESSING EMAS-3 FACILITIES

Fortran as implemented on EMAS provides access to all the intrinsic functions which are defined in the Fortran77 standard. They are listed in the Fortran77 Language Manual. If you write Fortran in order to produce portable programs then you should limit your use of external functions and routines to these (plus NAG routines if appropriate).

If, on the other hand, you wish to access EMAS facilities you may find some of the following routines and functions useful. Remember though that they are non-standard and you may have problems if you try to take your programs which use them to other systems.

9.4.1 Use of CHARACTER variables

Most of these routines and functions make use of CHARACTER variables. When you want to send text to EMAS then you can use either a constant or a variable. When you use a constant the effect is obvious, for example:

```
CALL EMAS3PROMPT('Next:')
```

will set the prompt for subsequent input from the interactive terminal to be `Next:`.

If, however, you use a variable then you must remember that if the value assigned to it is shorter than the variable it will be space filled on the right. For example:

```
CHARACTER HP*10
HP='Next:'
CALL EMAS3PROMPT(HP)
```

will set the prompt to the text `Next:` followed by 5 spaces.

9.4.2 Language Independent Interface

Earlier versions of EMAS have provided limited access to Subsystem facilities for Fortran users through a number of special Fortran interface routines. With EMAS-3 the program interface is language independent. All the procedures in the Subsystem which have names beginning with "EMAS3" are callable from Fortran. A full list of these interface routines is contained in User Note 80. The following sections give examples of some of the more commonly used ones.

9.4.3 Setting the prompt text

The routine EMAS3PROMPT is used to specify the text to be used when prompting for subsequent input from the interactive terminal. It has no effect during batch jobs. It takes one parameter of type CHARACTER. This can be a constant or a variable.

Example:

```
      CHARACTER PTEXT*10
      :
      :
      CALL EMAS3PROMPT('NEXT ROW:')
      :
      :
      PTEXT='AGAIN:'

C NOTE THAT PTEXT WILL BE SPACE FILLED AND
C THESE SPACES WILL BE PRESERVED IN THE PROMPT

      CALL EMAS3PROMPT(PTEXT)
```

9.4.4 Detecting Interrupts from the Interactive Terminal

Multi-character interrupts from the interactive terminal can be generated as explained in Section 14.6.1. They can be detected by using the routine EMAS3INTERRUPT. It takes one variable of type CHARACTER. When it is called a check will be made for an outstanding multi-character interrupt.

If there is an interrupt waiting, it will be put in the character variable (space filled or truncated as for assignment). If there is no interrupt outstanding the character variable is filled with spaces. Note that in the following example a comparison is made with a single space to determine whether an interrupt has been detected. This is allowed since trailing spaces are ignored in CHARACTER string comparisons.

```
CHARACTER INTHOLD*5
:
:
CALL EMAS3INTERRUPT(INTHOLD)
IF (INTHOLD.NE.' ') THEN
:
IF (INTHOLD.EQ.'STOP') THEN
:
:
GO TO 100
:
END IF
:
IF (INTHOLD.EQ.'RESET') THEN
:
GO TO 200
:
END IF
END IF
```

This routine provides an additional means of controlling interactive programs. It has no effect in batch jobs - it will always fill the character variable with spaces.

9.4.5 Calling EMAS commands

It is possible to call EMAS-3 foreground commands from within Fortran programs. The routine EMAS3 takes two parameters of type CHARACTER followed by an integer variable. The first is the name of the command and the second the parameter string for the command. The third parameter is used to return the result of the command itself. If this is zero then the command executed correctly.

Example:

```
CALL EMAS3('LIST','TEXTFILE,.LP15',IFLAG)
```

9.4.6 Checking the existence of a file

Sometimes it is useful to be able to check whether a file of a particular name exists. The subroutine EMAS3EXIST can be used for this purpose. It takes two parameters. The first must be a CHARACTER variable or constant which holds the name of the file. The second must be an integer. After a call of EMAS3EXIST the latter will be set non-zero if the file specified in the first parameter exists and is permitted to the calling user in at least one mode.

```
INTEGER FOUND
:
CALL EMAS3EXIST('EGNP80:DATA23',FOUND)
IF (FOUND.NE.0) THEN
:
:
```

Obtaining information about the Subsystem Environment

Two subroutines are available which provide information about the environment in which the program is running. EMAS3UINFI is a routine which returns an integer value - for example the number of users currently logged on; EMAS3UINFS is a routine which returns character information - for example the Username of the process in which the program is running. EMAS3UINFI and EMAS3UINFS have been

written in such a way that they can readily be extended to provide additional information.

9.4.7 The subroutine EMAS3UINFI

This takes two integer parameters. The first is used to specify which piece of information is required (see Appendix 5). The information required is returned in the second parameter.

Example:

```
            CALL EMAS3UINFI(3,NUSER)
            WRITE (6,100) NUSER-1
100         FORMAT (I4,' people are sharing EMAS with you')
```

9.4.8 The subroutine EMAS3UINFS

This takes two parameters. The first is an integer which specifies which item of information is required (see Appendix 5). The second is a character variable into which the information will be placed, (truncated or space filled as for assignment).

Example:

```
            CHARACTER NAME*20
            CALL EMAS3UINFS(7,NAME)
            WRITE (6,100)NAME
100         FORMAT (' Hello ',A20)
```

9.4.9 Calling IMP

The object files produced by the main EMAS-3 compilers are compatible and it is possible, within the limits imposed by the parameter passing facilities of the two languages, to make calls between object files of programs written in different languages. This is particularly useful to the Fortran programmer in that it makes it possible to exploit some of the more sophisticated facilities of EMAS-3. The mechanisms for making calls between Fortran and IMP are described in the Fortran77 Language Manual.

9.5 INPUT/OUTPUT

EMAS-3 provides the necessary support routines for the input/output facilities of the Fortran language. There are two main access methods: sequential and direct access, and both allow for the optional use of a FORMAT statement.

9.5.1 File types

Table 9.2 indicates which file types and devices can be used for each access method. The file types are described in Chapter 4.

	Sequential		Direct Access	
	With FORMAT	Without FORMAT	With FORMAT	Without FORMAT
Input	Terminal (.IN) Character File Data File Magnetic Tape	Data File Magnetic Tape	Data File	Data file
Output	Terminal (.OUT) Line Printer (.LP) Data File Character File Magnetic Tape	Data File Magnetic Tape	Data File	Data File

Table 9.2 Summary of Fortran I/O Access Methods

9.5.2 Use of DEFINE with Fortran

The command DEFINE is used to establish a link between a logical channel number in a Fortran program and a particular file or device. The command, and the associated command CLEAR, are described fully in Chapter 13. When used with Fortran the normal form of the command should be

```
      DEFINE n,filename
or    DEFINE n,device
```

For example:

```
Command: DEFINE 23,DATA107
Command: DEFINE 1,.LP
```

DEFINE can also be called within a Fortran program. For example:

```
CALL EMAS3('DEFINE','23,DATA107',IFLAG)
```

9.5.3 Sequential input

It is possible to use the interactive terminal (.IN), a character file, a data file or a magnetic tape for input (see Table 9.2 above). When using the interactive terminal or a character file the record is padded with blank characters up to column 80 if necessary. This facilitates the transfer of programs from card-oriented systems. For example, if it is required to read a title of up to 80 characters from the interactive terminal, the user has only to type the printable characters - the rest will automatically be filled with spaces.

Example:

```
      INTEGER TITLE(20)
100   FORMAT(20A4)
101   FORMAT(' ',20A4)
      CALL EMAS3PROMPT('Title:')
      READ(5,100) TITLE
      WRITE(6,101) TITLE
```

(Note the carriage control character in the FORMAT statement used for output.)

When run, this would appear on the interactive terminal (assuming default channel definitions) as:

```
Title:FIRST EXPERIMENT
FIRST EXPERIMENT
```

When a data file is being read, on the other hand, the length of the input record must be at least as long as that implied by the FORMAT being used. Otherwise the Fortran run-time fault

```
Wrong length record
```

will occur.

9.5.4 Sequential output

Sequential output written using FORMAT control can be directed to data files, character files, the interactive terminal, or various output devices (see Table 9.2 above). It is important to understand the effect of the carriage control character ('format effector') which is often put at the start of the output record. If the output is directed (by command DEFINE) to a device that can interpret this - for example, the interactive terminal or a line printer or a CHARACTER file - then it is interpreted as explained in the Fortran77 Language Manual. If, instead, it is directed to a DATA file, then it is treated as part of the output record. One of the effects of this is that if output intended for eventual printing is first put in a file and then listed by using the LIST command, the presence of carriage control characters must be indicated to LIST, so that the first character of the line can be interpreted correctly. For example:

```
          :
100       FORMAT('1HEADING OF OUTPUT')
          WRITE(8,100)
101       FORMAT('0',3I10)
          DO 1 I=1,30
  1       WRITE(8,101) LAB(I), KRY(I), NUM(I)
          :
          :
```

If the command

```
Command: DEFINE 8,.LP
```

were typed before running this program the output would be printed on the line printer, correctly interpreting the carriage control characters for 'newpage' and 'two new lines'. If, however, it were required to put the output in a file and then to list the file later, it could be defined thus:

```
Command: DEFINE 8,OUTLP,,C
```

Note that this would cause a character file to be created; if the C parameter were omitted a data file would be created. LIST can handle either.

9.5.5 Direct access files

When using direct access files the following points should be noted:

* Only files can be used, not devices.

* .NULL (the dummy file specification) cannot be used.

* The size and record length of the file are not taken from the DEFINE command:

they are taken from the OPEN statement in the Fortran program used when the file is first referenced.

9.5.6 Default definitions

When running Fortran programs the following definitions are established by default; they can be overridden by use of the DEFINE command.

Channel	Input/Output	Device	
		Foreground	Background
5	Input	.IN (user's terminal)	.IN (Job file)
6	Output	.OUT (user's terminal)	.OUT (Job output file or device as defined as LOG in DETACH)

Table 9.3 Fortran Default Channel Definitions

Note that the meanings of .IN and .OUT depend upon whether the program is running in foreground or background mode. Note also that the effect of channel 5 being defined as the Job file is that the data to be read on channel 5 can follow immediately after the RUN command in the Job file (see Chapter 16). For example, the following program, compiled into object file FORT73, could be run by using the Job file shown:

```
        Program                          Job file
        :                                :
        :                                :
100     FORMAT(2I4)                      RUN FORT73
        READ(5,100) NUM,TOP               276 389
        :                                :
        :                                :
```

9.5.7 Effect of OPEN on definitions

All connections set up by OPEN statements are terminated on completion of the run of the program.

A stream may still be set up prior to the run of the program by using DEFINE, and actually opened within the program by using OPEN. In this case the definition is seen to have been set up outwith the Fortran environment, and will remain in force on completion of the program run.

If FILETYPE='CHARACTER' or FILETYPE='C' is specified in an OPEN statement then the file being created will be of type character even if the FORMAT parameter of the DEFINE command is not used to specify a file type other than the default of DATA.

CHAPTER 10
PASCAL ON EMAS-3

Information about compiling and executing Pascal programs on EMAS-3 was not available when this Guide was written, but User Note 64 contains some information.

A call of the command

```
PASCAL source, object, listing, errors
```

will produce an object file analogous to one produced by the commands IMP and FORTRAN (see Section 7.2.1).

CHAPTER 11
THE C PROGRAMMING LANGUAGE ON EMAS-3

Information about compiling and executing C programs on EMAS-3 was not available when this Guide was written.

CHAPTER 12
OTHER PROGRAMMING LANGUAGES ON EMAS-3

12.1 INTRODUCTION

Chapters 8 to 11 of this Guide provide information about the use of IMP80, Fortran77, Pascal and C on EMAS-3. Consideration is being given to implementing a number of other programming languages to make it easier for you to import programs from elsewhere, and to allow you to work with languages you have used on other systems. The choice of which languages to implement will be influenced by requests from users.

12.2 DESIGN AIMS FOR PROGRAMMING LANGUAGES ON EMAS-3

In implementing programming languages on EMAS-3 there are a number of aims:

* to use a version of the language which conforms to an International standard where such a standard exists.

* to provide compilers which produce standard object files which can be loaded by the standard Subsystem loader (see Chapter 7). This ensures that when you use a compiled program you do not need to be aware of the programming language used.

* to support cross calling between procedures written in different languages - subject to the availability of suitable parameter structures within both languages.

* to allow programs written in any language to make a wide range of calls on the Subsystem interface.

12.3 PROGRAM PORTABILITY

EMAS provides an excellent environment for developing programs. If you use EMAS to develop programs you should consider their eventual use. Often programs which start out with limited aims are extended over the years to provide a much wider range of functions than was initially envisaged. In their developed form they may become useful to colleagues working in other institutions. If you think your programs are likely to develop in this way then the following points may assist your planning:

* In the case of commonly available languages, e.g. Fortran and Pascal, you should avoid using features of the language outwith the standard. The compilers contain facilities to help you identify the use of such features.

* You should be wary of using EMAS specific procedures. Some, such as date and time routines are likely to be available on most systems in a form similar to those provided on EMAS. Others, such as those concerned with prompting or multi-character interrupts are unlikely to be available.

* At the very least you should include explanatory comments at all places where your program calls EMAS specific procedures. This will assist the person who has to convert the program to run on another system.

* Store Mapping (see Chapter 19) is not generally available outwith EMAS.

* If you need to transfer data to another system, either via magnetic tape or by using the TRANSFER command (see Section 15.10) you will find it much easier if the data is held in character, rather than binary format. For programs which handle large quantities of numeric data, it is usually more efficient to use binary files (e.g. Fortran unformatted files). If this is the case, you will need to write a pair of simple programs to convert the data to and from character format. The first will be run on EMAS. The second will be run, after transferring the character data, on the other system.

If you anticipate the need to write substantial programs for running on EMAS and another system you should seek advice at an early stage - both from the ERCC Advisory service, and from the Advisory service responsible for the other system.

CHAPTER 13
PROGRAM ACCESS TO DATA

13.1 INTRODUCTION

Whilst some programs obtain all their data from the interactive terminal and output all their results to the terminal, many need to read from data stored in files. Others will need to write their output into files for long term storage or subsequent analysis by other programs, or to write to output devices such as printers or plotters. This chapter describes the structure of the files used to hold data on EMAS and the mechanisms provided for linking your program to particular files or devices.

13.2 Files used for holding data

There are two file types used for holding data on EMAS:

* CHARACTER files are normally used to hold information as printable characters.

* DATA files are normally used to hold binary information. This is a direct copy of the contents of variables in your program.

Standard EMAS files such as CHARACTER files and DATA files are made up of two parts:

* The first part is a header, usually 32 bytes long, which contains information about the file's type, size and structure. It also contains a record of when the file was last altered.

* The rest of the file contains the data. The descriptions below refer to the layout of the user data area only. The header is not normally accessed directly by user programs. Its contents are maintained by the Subsystem, and can be displayed by using the command ANALYSE (see Section 4.3).

The EMAS file system relieves you of any need to be concerned about the way your files are stored; in particular you do not need to know about the size of the blocks on the disc.

13.3 CHARACTER files

CHARACTER files are used widely on EMAS. They can be created and modified by using the editors EDIT and ECCE. They are used to hold the source files for all the programming languages, and for any other application involving textual information. They are generated by the compilers when a compilation listing is produced, and can be created by commands such as FILES, when you require a list of filenames to be placed in a file.

Although CHARACTER files are normally used to hold textual information made up of sequences of printable characters, they can be used to hold any byte values in the full range 0 - 255. This means that they can be used to hold data for special devices which use values outwith the EMAS internal code (see Appendix 3).

CHARACTER files have no structure imposed on them by EMAS. However, for most

purposes they contain printable characters separated into lines by the EMAS line separator character NL which has an internal code value of 10. Files which conform to this convention can be listed correctly on your interactive terminal or on a line printer, subject only to the requirement that each line is not longer than the maximum allowed for the device being used. The only control characters other than NL that normally appear in such files are:

FF (Form Feed) - this is the character which causes the line printer to start a new page. It has an internal code value of 12.

CR (Carriage Return) - this is used to move back to the beginning of the current line in order to allow overprinting. This can be used for underlining, for example. If a CR character is adjacent to an NL character then it has no significance, since NL is interpreted as 'move to the beginning of the next line'.

As a simple example a CHARACTER file containing the text:

```
FIRST LINE
END
```

in its only two lines would contain 15 characters of data, including the two NL characters. The information would be stored in the file thus:

	"F"	"I"	"R"	"S"	"T"	" "	"L"	"I"	"N"	"E"	nl	"E"	"N"	"D"	nl
byte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

13.4 DATA files

DATA files on EMAS are used to hold all types of information. They can be used to hold textual information, though they are less suitable than CHARACTER files for this. They can also be used to hold binary information, a direct representation of the information contained in the variables of a program. The records in a DATA file can be in one of three different formats:

F - Fixed length records

V - Variable length records

U - Unstructured - where the file can be considered as being one record.

13.4.1 The structure of FIXED format DATA files

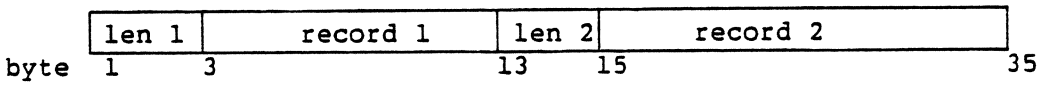
All the records in a Fixed format DATA file are of the same length. The record length is measured in bytes. So, for example, a DATA file with a format of F20 contains a series of 20 byte records. The records are contiguous within the file, with no intervening markers. For example the F20 file would be laid out thus:

	record 1	record 2	record 3	record 4
byte	1	21	41	61

F format files use file space efficiently and are ideal for applications which hold many items of information of the same size. They can be used for sequential access (reading or writing records in order). For direct access (reading and writing records in a random order), it is required to use fixed format files.

13.4.2 The structure of VARIABLE format DATA files

Variable format data files allow for records of different sizes to be mixed within a file. The format of such a file specifies a maximum record length. For example a file with a format V1024 can contain a mixture of records varying from 1 to 1024 bytes in length. Each record is preceded by a two byte length field. This contains, in binary representation, the length of the record (including these two bytes). For example, if a file contained two records of 10 and 20 bytes respectively its structure would be:



V format files are ideal for many applications, although they cannot be used for Direct (Random) access.

The overhead imposed by the inclusion of the length fields is significant only if you use very short records.

13.4.3 The format of UNSTRUCTURED DATA Files

An unstructured DATA file, as its name implies, has no structure imposed on it by the Subsystem. It is best thought of as one record, with length equal to the length of the whole file. Unstructured Data files are accessed via Store Mapping (see Chapter 19).

13.5 Linking programs to files and devices

Most programs do not have the names of the files they access built into them. Instead they access files and devices via logical channels. A channel is identified by a channel number (normally an integer - but see language chapters for details of channel identifiers for each language).

Before a program can access data via one of these logical channels you must assign the logical channel to a particular file or device. There are two methods of doing this:

* By using the command DEFINE. This is the normal mechanism for user written programs.

* By using the facility for setting up a `route`, provided by the Parameter Acquisition Mechanism (Section 18.7). This method is used by many commands which access files, and you can use it if you write your own commands.

13.6 The DEFINE Command

There are some slight differences in the way DEFINE operates depending upon which language the program is written in. You should also read the appropriate language chapter for this information.

```
DEFINE channel, file or device, size, format
```

13.6.1 The CHANNEL parameter

This relates to the identification of the logical channel in the program. In the case of IMP and Fortran this is normally a positive integer. For example if your Fortran program contains the line:

```
120   WRITE(70,100) NCOUNT, NAR, NFR
```

Then you would need to define channel 70.

If DEFINE is used for a channel which is currently defined then the old definition is lost and replaced by the new one.

You are advised to use channel numbers not greater than 1000. Numbers above 1000 may be used by the Subsystem. See EMAS3CLAIMCHANNEL, Section 13.10.

13.6.2 The FILE or DEVICE parameter

This parameter is used to specify the file or device to be associated with the channel. Using the example above you could type:

```
Command: DEFINE 70,MYDATA18
```

If you choose to send the output to the line printer you would type:

```
Command: DEFINE 70,.LP
```

Further information about output devices is given in Chapter 15.

Other options available for this second parameter are:

`Command: DEFINE 70,.OUT`	output to the interactive terminal or to the Job journal if a batch job.
`Command: DEFINE 70,.TEMP`	output to a temporary file. It will remain available until you log off or re-DEFINE channel 70
`Command: DEFINE 70,.NULL`	this is a useful way to dispose of output not required during this run.

If the channel being DEFINEd is to be used for input then the only device available is .IN, the interactive terminal (or job file if a batch job).

```
Command: DEFINE 65,.IN
```

Other options are:

`Command: DEFINE 65,DATA.YEAR1986`	read from a file.
`Command: DEFINE 65,.TEMP`	read output previously written via this logical channel using `.TEMP`
`Command: DEFINE 65,.NULL`	returns `'Input Ended'` if an attempt is made to read from this channel.
`Command: DEFINE 65,DATAJAN+DATAFEB`	concatenated input. Files must be of the same type and format.

13.6.3 The SIZE parameter

The size of the file has different meanings according to the method by which the file is being used:

* If you are only reading from the file then this parameter is ignored.

* If the file is to be used for sequential output then this parameter specifies the maximum size the file will be allowed to reach during this definition.

* If the file is to be used for Direct Access and already exists then this parameter is ignored.

* If the file is to be used for Direct Access and does not exist then the effect of this parameter depends upon the programming language being used.

* If the file is to be used for Store Mapping then this parameter is ignored.

The size can be expressed as an integer, in which case it is interpreted as a number of Kilobytes (units of 1024 bytes). For example, if you wish to restrict the output of a program to 10 Kilobytes then you could use the command:

```
Command: DEFINE 70,TEMPOUT,10
```

Note that the output may not stop when the file reaches exactly 10 Kbytes. The size is actually rounded up to the next multiple of 4096 bytes, having allowed for the length of the header.

An alternative way to specify the size is in records. This method can only be used when you specify an F format (see below). For example if you specify that the file is to have F100 (100 byte) records then you could specify the size as R50.

```
Command: DEFINE 60,DBASE,R50,F100
```

This would tell the Subsystem to allow room for 50x100 bytes of data, i.e. room for 50 records.

Note that you will not often need to specify the size parameter since the default value will serve for many applications.

13.6.4 The FORMAT parameter

This parameter is used to specify the format. It is only relevant for sequential output files. The three formats for DATA files are fully described earlier in Section 13.4. Typical values for this parameter would be:

F80 - fixed length records of 80 bytes each

V10000 - Variable length records, with a maximum of 10000 bytes.

Note that this parameter has no effect when you are using Direct Access or Store Mapping to access the contents of the file. Under some circumstances you may wish to ensure that the output file is a CHARACTER file. In this case you should set this parameter to 'C' (see Section 9.5.4):

```
Command: DEFINE 34,TEXTFILE,,C
```

13.6.5 Listing current definitions

To find a list of current definitions you should type:

```
Command: DEFINE =
```

13.7 Accessing data on magnetic tape

Chapter 21 explains how to access data on private magnetic tapes, and describes the DEFINEMT command which is a direct equivalent of DEFINE.

13.8 Calling DEFINE from within a program

To avoid having to type the DEFINE command each time you use a program you may find it more convenient to make the call from within the program. To do so you should use the routine EMAS3. The way to use this from a particular language is described in the appropriate language chapter. As a simple example you could DEFINE channel 50 to access a line printer in IMP and Fortran respectively by using the following statements:

```
EMAS3("DEFINE","50,.LP",FLAG)          for IMP

CALL EMAS3('DEFINE','50,.LP',IFLAG)    for Fortran
```

13.9 The command CLEAR

This command is used to remove definitions set up by DEFINE. It can be used with one or more channel identifiers. Alternatively it can be used with no parameter in which case all definitions will be cleared.

```
Command: CLEAR 50          channel 50

Command: CLEAR 60&70       channels 60 and 70

Command: CLEAR             all currently defined channels
```

13.10 Choosing a unique channel number

If you build fixed channel numbers into your programs then there is a danger that they may conflict with other programs or commands which you use. For example if your program outputs its results on channel 70 you will have to call DEFINE for channel 70 before you run it. If you later use a command belonging to someone else it may re-use channel 70 for some other purpose, unknown to you. Unless you check the definition of channel 70 before re-running your program you may overwrite one of your files inadvertently.

This problem can be circumvented by use of the routine EMAS3CLAIMCHANNEL. This routine returns a channel number which you can safely use with no chance of it affecting other definitions. In the following example an output channel is needed which is defined to be a line printer. Here is the IMP code:

```
%EXTERNALROUTINESPEC EMAS3CLAIMCHANNEL(%INTEGERNAME CHAN)
%EXTERNALROUTINESPEC EMAS3ITOS(%INTEGERNAME I,%STRINGNAME S)
%EXTERNALROUTINESPEC EMAS3(%STRINGNAME COM,PARAM,%INTEGERNAME FLAG)

%INTEGER CHAN,FLAG
%STRING(15) SCHAN

   EMAS3CLAIMCHANNEL(CHAN)
   EMAS3ITOS(CHAN,SCHAN)
   EMAS3("DEFINE",SCHAN.",.LP",FLAG)
   SELECTOUTPUT(CHAN)
   PRINTSTRING("Heading on output")
```

And here is the equivalent Fortran:

```
      INTEGER CHAN,FLAG
      CHARACTER*10 SCHAN
      CALL EMAS3CLAIMCHANNEL(CHAN)
      CALL EMAS3ITOS(CHAN,SCHAN)
      CALL EMAS3('DEFINE',SCHAN//',.LP',FLAG)
      WRITE(CHAN,100)
100   FORMAT(' Heading on output')
```

CHAPTER 14
INTERACTIVE TERMINALS

14.1 Introduction

The primary method of using EMAS is via an interactive terminal. This chapter explains how an interactive terminal communicates with EMAS, and how you can use your interactive terminal most effectively.

14.2 The connection between your terminal and EMAS

In the early days of EMAS, interactive terminals were connected directly to the mainframe computer on which the EMAS system was running. Later an intermediate computer, called a Front End Processor (FEP) was connected to the mainframe, and the terminals were connected to the FEP. Now the FEP no longer communicates with individual terminals, instead it is connected to a communications network. In the case of EMAS this is a network called EDNET which links together several computer systems (known as Host computers).

Groups of terminals are connected to small computers known as PADs (see below), which are, in turn, connected to EDNET. The predecessor of the PAD was known as a TCP (Terminal Control Processor). At the time of writing a few of these are still in service, but they are being phased out. In many respects, using a terminal connected to a TCP is the same as using a terminal connected to a PAD, but there are minor differences. These differences are not covered in this Guide. If you are using a terminal connected to a TCP you should read User Note 52.

14.2.1 PADs

A PAD, strictly a Packet Assembler/Dis-assembler, is a small computer to which a group of typically 16 interactive terminals is connected. The PAD is also connected to the communications network which gives the users of the interactive terminals access to a number of Host computers. The name of the PAD is derived from its functions. Information is passed around the network in Packets - blocks of characters. The job of the PAD is to receive individual characters from the terminal, Assemble them into Packets and send them via the network to the Host concerned. Output from the Host, directed to the terminal, arrives via the network at the PAD, which Dis-assembles it into individual characters for sending to the terminal.

14.2.2 Terminals connected via modems

Most terminals are connected directly to PADs. Some are connected via a telephone line using a device called a modem. The modem converts the characters generated by the terminal into audible tones suitable for sending over the telephone network. At the far end of the line is another modem which is connected to a PAD. The details of how to use a particular modem are outwith the scope of this Guide. If you are using a terminal connected via a modem and you experience any problems you should consult the person responsible for the terminal and modem. Once you have established communication with the PAD you should continue exactly as for users whose terminals are connected directly to the PAD.

14.2.3 Terminals connected to other networks

If your terminal is connected, via a PAD or other form of terminal concentrator, to a network other than EDNET, you may still be able to access EMAS if your network is connected directly or indirectly to EDNET. You will need some information about how to make a call to EDNET which should be available from the administrator of your network. Thereafter you should be able to proceed as if your terminal was connected to a PAD on EDNET. Although there is a standard for the operation of PADs, not all suppliers and network implementors have interpreted the standard in the same way. This means you may find minor differences from the description here. Again, such differences are outwith the scope of this Guide: User Note 52 may be of help. Also you should read the documentation for the PAD you are using.

14.3 Controlling the PAD

Whilst most of the time you will be using your terminal to communicate with EMAS or another Host, you can also communicate directly with the PAD. You must do this when you log-on to EMAS because you have to tell the PAD which Host you wish to use (see Section 2.3.2). You can do it at other times if you wish to change some aspect of the PAD's behaviour, or if you wish to generate a terminal interrupt (see below).

14.3.1 Direct control of the PAD

before logging on, you should press the <RETURN> key a few times to initiate communication with the PAD. You should see the name of the PAD displayed followed by the prompt:

```
PAD>
```

Once logged on, the method is to type <CTRL+P> (hold down the <CTRL> key and press 'P'). This switches communication from the Host to the PAD. Next you should type 'A' which again results in a prompt:

```
PAD>
```

You can now type a command which will be interpreted by the PAD. The details of these commands can be found in the PAD documentation, of which a summary sheet should be available near the terminal.

14.3.2 Control of the PAD using the SETMODE command

A more convenient method of controlling the PAD is to use the SETMODE command. This is an EMAS command which can be used at command level, or which can be called from within a program by using the EMAS3 procedure. It sends instructions to the PAD without you having to enter into a direct dialogue with the PAD as described above. There are a number of keywords, which are described in User Note 52. Many are fairly abstruse and are unlikely to be required. The following are of general use:

MODE - this can take one of four values:

MESSAGE - this is the standard operating mode when using EMAS

NATIVE - forwards input without waiting for you to press <RETURN>. Required by some screen editors.

TRANSPARENT - avoids "line-folding" of long lines on output. The PAD does not interfere with the data sent by the Host.

GRAPH - like TRANSPARENT, but in addition prevents the Host from adding a CR character to each LF output.

ECHO/NOECHO - by default all input is echoed on the terminal. Use SETMODE NOECHO to stop this. This mechanism is used during log-on when your password is being read.

CONTROLCHARINTS - see section on generating interrupts below.

The form

```
Command: SETMODE =
```

can be used to obtain information about the current settings. You will need User Note 52 to interpret the output.

A call of SETMODE with no parameters returns the PAD to the mode settings in force at the start of the interactive session.

As with all EMAS-3 commands, the call SETMODE ?? provides help information on the whole command, and a response of "?" to a prompt gives information about the item being prompted for.

14.4 Telling EMAS about your Terminal

One of the unfortunate characteristics of interactive terminals is that they do not conform to any one standard in the precise way they operate. In particular there is no standard way to send control signals to perform functions like clearing the screen. For many operations these differences do not matter but for some, particularly the VIEW and HELP commands and any screen editors, these differences do matter. For this reason you should use the command TERMINALTYPE to tell EMAS what sort of terminal you are using.

To use the command just type:

```
Command: TERMINALTYPE
```

You will then see a list of terminals and will be invited to select one by typing its number, followed by <RETURN>. If your terminal is not on the list you will be able to get by initially by using one of the early numbers which are general purpose descriptions - e.g. "Unspecified Video" or "Hard Copy Width 80". If you expect to make regular use of EMAS using this terminal you could ask the Advisory service to arrange for it to be added to the list recognized by TERMINALTYPE.

To save you having to type TERMINALTYPE every time you use EMAS the Subsystem makes note of the address of the terminal you are using and the type you have specified. It can hold several such mappings of addresses and types so it can cope with you logging-on from several different terminals, so long as you use TERMINALTYPE once at each terminal. Each time you log on you will see a message saying what type of terminal the Subsystem believes you are using. If this is wrong then you will have to use TERMINALTYPE again. This situation will normally only arise if a different type of terminal has been plugged in to a PAD socket which you have used before.

14.5 OPTION settings related to Interactive Terminals

The OPTION command is described in general in Section 17.9. Several OPTION parameters relate to the interactive terminal:

BLANKLINES - this can take values ON (the default) or OFF. If OFF is selected then completely blank lines are not displayed on your terminal. This results in more information being displayed at one time, with some loss in clarity.

CONTROLCHARINTS - see section on generating interrupts below.

DEFAULTTERMINALTYPE - the mechanism whereby the Subsystem stores a list of mappings between terminal addresses and types is not foolproof because the network is under constant development and terminals are sometimes moved. You can use this parameter to specify what type of terminal you wish to default to if the Subsystem cannot work out what type of terminal you are using. The value for this parameter should be that used for TERMINALTYPE. For example:

```
Command: OPTION DEFAULTTERMINALTYPE=22
```

will cause the Subsystem to assume that you are using a Hazeltine Esprit II visual display unit if it cannot determine the type from its list of known terminals. You need not type the complete option name - the abbreviation `DEF=22` is adequate.

ITINSIZE and ITOUTSIZE - can be used to specify the sizes of the interactive terminal input and output buffers in the Subsystem. These are specified in Kbyte. For example:

```
Command: OPTION ITINSIZE=8
```

sets the size of the input buffer to be 8 Kilobytes. See also Section 15.3.1.

RECALL - see Section 14.7 below on the interactive journal.

14.6 Interactive Terminal Interrupts

At times you may wish to alter the progress of a program or command whilst it is still executing. For example you may have LISTed a long file on the interactive terminal and then realized that you do not wish to see it all. There are two types of interrupt. Single character interrupts are detected by the Subsystem and their uses are fixed. Multi-character interrupts can be detected by your program and used to change its operation whilst it is running.

14.6.1 Standard method of generating interrupts

The standard method is to type <CTRL+P> (hold down the <CTRL> key and press 'P'), and then press 'B'. The result will be a prompt:

```
Int:
```

You should reply with a single or multi-character interrupt as required. Note that if you generate an `Int:` prompt by mistake you should just press <RETURN> and the interrupt will be ignored.

14.6.2 Shorthand method of generating some single character interrupts

For the single character interrupts A, C, K and T there is an alternative and faster method of generating interrupts. All that is required is that you hold down the <CTRL> key and press the required letter. This is not only simpler to type but messages sent by this means are detected more quickly by the Host computer which runs EMAS.

The only problem with this mechanism is that under some circumstances you may wish to pass characters such as <CTRL+A> through to your program. In this case you can type SETMODE CONTROLCHARINTS=OFF to disable this facility. If you regularly use characters such as <CTRL+A> among your input you can make this your default by using the OPTION CONTROLCHARINTS=OFF.

14.6.3 Meanings of single character interrupts

Interrupt	Effect
A	Abort program and return to Command level.
C	Same as A but also discard any input that has been typed ahead.
K	Discard all terminal output until the next prompt for input. Note that this interrupt does not affect the running program, it is only its output which is suppressed.
Q	Abort current command or program, print diagnostics and return to Command level.
T	Print CPU time in seconds and number of pageturns since the start of the current command, together with the number of users logged on, all without affecting the running program.

Table 14.1 Single character interrupts

14.6.4 Multi-character interrupts

These can be detected in your program by using the EMAS3INTERRUPT procedure which is described in the chapters about each programming language (Sections 8.6.3 and 9.4.4). You can use this mechanism for controlling programs which run for long periods without reading data from the interactive terminal.

14.7 Reviewing your terminal conversation

Although in many respects visual display units are superior to hard copy terminals – for example they operate at higher speeds and are much quieter, they do not provide an intrinsic method of looking back over earlier work. For this reason EMAS contains a mechanism to create a journal of your interactive terminal transactions. This mechanism is controlled by the OPTION command:

OPTION RECALL=PERM	preserve journal from one session until the next. This is the default setting.
OPTION RECALL=TEMP	preserve journal for current session only.
OPTION RECALL=NONE	do not record transactions at all.

The information is stored in a circular buffer. If it becomes full then the earliest material is overwritten by the most recent transactions. To interrogate the information in the journal you should use either RECALL (see Section 5.9) or RECAP (see Section 5.21).

14.8 Messages from other processes

Apart from messages relating directly to the commands you use and the programs you run, you will sometimes receive messages from other EMAS processes on your interactive terminal. For example:

* A message from the operators warning you that a session is about to end.

* A message to say that there is new MAIL for you to read (see Section 15.12).

* A message to say that a file you have requested by using RESTORE has arrived (see Section 3.9.3).

* A TELL message from another user (see Section 14.10).

These messages are of the form:

```
**VOLUMS 06/07/86 15.24 CHAP27S restored
```

The source of the message will normally be the name of the process from which it originated. Messages relating to the ARCHIVE or BACKUP stores come from VOLUMS; messages about mail come from MAILER and messages from other users will be identified by their usernames.

14.9 Suppressing Messages

At times such messages can interfere with your output, for example if you are running a program which has output which is designed to fit the size of your screen exactly. In this case you can use the command MESSAGES to suppress these messages. The command takes one parameter which should be OFF or ON.

Use `MESSAGES OFF` to suppress this type of message. Messages from the operators will be lost completely. This means you will get no warning that the session is about to end. Other messages will be stored. They will all be printed as soon as you type `MESSAGES ON`. If you do not type this before you log off then the messages will be printed as soon as you next log on.

14.10 The TELL command

This command provides a mechanism for sending short messages to other users. If the message takes more than a few lines you will probably find MAIL a more suitable command for sending it (see Section 15.11).

The simplest way to use the command is to use one parameter, the username of the recipient of the message. Then type the message, terminated with a line containing a single asterisk. For example:

```
Command: TELL ERCN19
Tell:New version of program in MYSS03Y
Tell:*
```

As an alternative you can create a file to hold the text of the message by using an editor such as EDIT or ECCE. You should then type the name of this file as the second parameter to TELL:

```
Command: TELL ERCN19,MESSAGE
```

After typing the command you will get a response saying that the message has been sent or stored. In the former case the response indicates that the recipient is logged on and in the latter case the message will be stored until he or she next logs on. Even in the former case you cannot assume that the recipient has read the message since MESSAGES OFF (see above) may delay its output on the terminal.

CHAPTER 15
FILE INPUT AND OUTPUT

15.1 INTRODUCTION

Much of this Guide is concerned with EMAS-3 files - creating them, editing them, accessing them from programs and so on. This chapter explains how to get data into EMAS files from outside and how to list the data which is in your files on various output devices - e.g. line printers and graph plotters. It also explains how to transfer files and electronic mail between EMAS and other computer systems.

15.2 Card and Paper Tape Input

Punched cards and paper tape are rarely read directly into EMAS now, but facilities do exist to allow you to do this if you still need to. If you are accessing EMAS from within Edinburgh you should contact the Operations Manager or the Advisory service for details. You will need to submit your cards and paper tape to the operators together with:

* your username;
* your background password;
* the name you wish to give the file which is to contain the data from your cards or paper tape.

15.3 Input from a Microcomputer

A more common method of inputting data involves preparing it on a microcomputer by using a word processing package or some other program. You will then need to connect the microcomputer to EMAS as an interactive terminal. To do this you will need a suitable interface on the microcomputer, a cable to connect it to a communications port and a terminal emulator program. This is a program which runs in the microcomputer and makes it work like an interactive terminal.

As well as making the microcomputer work like an interactive terminal such programs usually incorporate facilities to transfer complete files between the microcomputer and the mainframe system, in this case EMAS. ERCC distributes two such programs for use on a wide range of computers. They are described briefly in the next two sections.

15.3.1 XTALK

XTALK has been developed at ERCC for a number of years. It is available for a wide range of microcomputers, particularly those which are widely used in the University of Edinburgh. If you just wish to use your microcomputer as an interactive terminal then you only need the part of XTALK which runs in the microcomputer. If you wish to transfer files between your microcomputer and EMAS-3 you will need access to the XTALK programs which run on EMAS-3 as well. To get access to these you will need to use the command:

```
Command: SEARCHDIR MICROS:EMASLIB
```

Note that you do not need to know the details of this software. It is accessed automatically when you use the file transfer facilities on your microcomputer. You will need to increase the size of your interactive terminal input buffer by using the command:

```
Command: OPTION ITINSIZE=16
```

You need only type these two commands once: they remain in effect between sessions. XTALK provides a reliable and efficient method of transferring data between your microcomputer and EMAS. For most situations it will be the best choice for this job, but in some cases you should use the alternative program KERMIT.

Further information about XTALK is contained in User Notes 74 and 91.

15.3.2 KERMIT

In contrast to XTALK which is primarily an ERCC program, KERMIT originated at the University of Columbia, New York. Its specification has been widely publicized in order to allow anyone to implement a version for their own computer. As a result versions of KERMIT are available on very many computers, both microcomputers and mainframes. KERMIT offers terminal emulation and file transfer facilities which are similar in scope to those provided by XTALK. There is a version of the KERMIT software on EMAS-3 and you can gain access to it as explained for XTALK above. KERMIT is less efficient than XTALK, and file transfer will be considerably slower. It is, however, much better able to provide reliable file transfer over noisy communications links. If, for example, you use EMAS via a telephone circuit you may find that you get corruption due to line noise. KERMIT is also the package to use if you access EMAS via a complex route - perhaps from a network which is only connected indirectly to EDNET. Information about KERMIT, including a list of the many computers for which it is available, is contained in User Notes 78 and 79.

15.4 Inputting data from magnetic tape

Chapter 21 explains how to read data on magnetic tape into EMAS. This is reasonably straightforward if the tape conforms to the IBM hardware and software standards which EMAS has adopted. If it is written in another format you may be able to read it on EMAS. An alternative route would involve reading it into a file on a machine of the type on which it is written, which itself has network access to EMAS. You would then need to use the TRANSFER command (see Section 15.10) to transfer the file to EMAS.

15.5 Inputting data from other Computer Systems

If you have data on another computer system you have several possible ways of moving it to EMAS. The most convenient method, if it is feasible, is to use the TRANSFER command (see Section 15.10) which uses the data communication facilities to move the data. EMAS is able to transfer data to and from many university and Research Institute computer centres, both in the UK and abroad. The TRANSFER command itself contains a facility to find out whether a particular system is accessible for file transfer.

An alternative method of moving data from another computer system is to use a magnetic tape. You should read Chapter 21 before you embark on this.

15.6 Terminology - Documents and Queues

When you direct a file to a device, either by using one of the commands described here or by DEFINEing a program output channel to a device, the Subsystem creates a Document. This consists of the information which is to be printed (or plotted) together with a descriptor which contains details such as the date and time at which the file was sent. This Document is put in a Queue. The name of the Queue will be the name of the device it is serving.

The same terminology is used for files awaiting transfer to a remote computer system and DETACHed jobs waiting to be run (see Section 16.3).

15.6.1 Finding out about your Documents

Use the command DOCUMENTS to find out about all your queued documents; those queued for an output device, awaiting execution, or awaiting transfer to another computer system. Just type:

```
Command: DOCUMENTS
```

You will be given a summary of the position and told which commands to use to find further information about particular types of queued documents.

15.6.2 Output Device Names

The names of output devices are constructed from three elements:

* A full stop;
* An abbreviation for the device type (e.g. LP for Line Printer);
* A numeric code or a textual abbreviation to identify the location of a device. In the case of devices connected directly to the computer system you are using the location is null (see example).

Examples:

.LP - local line printer attached to EMAS computer which you are using.

.GP23 - Graph Plotter number 23

.LPECHEM - A departmental Line Printer in the Edinburgh University Chemistry Department.

A full list of devices which are available can be found by using the command `HELP DEVICES.`

15.7 Outputting your files

There are three general commands for listing files to output devices. They are particularly intended for use with line printers and other devices which do not require special software to drive them. There are also a number of special commands for listing files to less standard peripherals such as graph plotters and laser printers. Normally it is best to use a special purpose command if one exists for the device you wish to send your file to. The following table lists the commands and explains where to find further information.

Command	Use	Details
LIST	Lists text files on interactive terminal or on any output devices accessible directly from the host	15.7.1
SEND	Similar to LIST but fewer options and destroys file.	15.7.2
NLIST	Similar to LIST but used for output to devices not directly accessible from the host	15.11.1
GPLIST	Graph Plotter output command. Deals with conversions required for many types of plotter.	User Note 17
DPLIST	Document Printer output command. Provides some document formatting functions.	User Note 48

15.7.1 The command LIST

```
LIST file, outdev, copies, forms, option
```

This command takes an obligatory parameter and four optional ones:

`file` The name of the file or member being listed; it can belong to this user or, if suitable access permission has been granted, to another user. Additionally, a concatenation of files may be given, by using the concatenation operator "+"; see the examples below.

`outdev` The abbreviated name for the device. The default is .OUT (normally the interactive terminal). Local devices or some of those connected via EDNET can be specified. Use the command `HELP DEVICES` for details of available devices.

`copies` This parameter can be used to specify the number of copies to be listed. If given, it must be an integer in the range 1-15. The default is 1.

`forms` This parameter can be used when listing files on output devices which permit special forms printing. The parameter is given as a number in the range 0-255. Contact the ERCC Advisory service for details of the forms codes available.

`option` If the file is a data file containing a format effector (carriage control character) at the start of each record, then this parameter should be specified as FE; otherwise it is not required.

The LIST command can be used to list character or data files on character or binary devices, but see note about special devices above. If a file is made up of bytes of user data which include newline characters where necessary but is unbroken by any control information, then it can be passed unmodified to an output device. Otherwise it will be converted to this form by LIST. (Character files and data files are described in detail in Sections 13.3 and 13.4.)

Examples

```
Command: LIST MND1136
Command: LIST ALIST,.LP
Command: LIST ERCC06:FILELIST,.LP15,,,FE
Command: LIST MYPROG+YOURPROG+ERXY99:ANOTHER.PROG,.LP23,2
```

LIST does not access devices directly, other than the user's terminal. Instead it makes a copy of the file, modifying it if it is a data file, and sends it to the SPOOLR process to be listed when the required output device is available. This may be minutes or hours after issuing the command. The command OUTPUT (see Section 15.8.1) can be used to determine whether any of your files are still waiting in the output queues. The command DELETEDOC (see Section 15.9.1) can be used to remove documents from these queues.

15.7.2 The command SEND

```
SEND file, outdev
```

The command SEND is similar to LIST but is more efficient, since it does not make a copy of the file being listed. Instead, it sends the file itself, unmodified, to the SPOOLR process. Thus the file is effectively destroyed by this command. When a file is not required other than to produce a listing, this command should be used.

Note that SEND does not modify the file in any way before passing it on to the SPOOLR process. Thus data files which would require the insertion of newline characters or the removal of record headers cannot be output by using SEND.

Notes

* SEND uses T#LIST as the default file, and .LP (line printer) as its default output device; hence

  ```
  Command: SEND
  ```

  (no parameters) causes the default compiler listing file to be output on the line printer and destroyed.

* SEND cannot be used for listing a file on the interactive terminal (.OUT).

* The `file` parameter cannot be a member of a partitioned file (cf. LIST).

* The `file` parameter cannot be a concatenation of files (cf. LIST).

* There are no `copies` or `forms` parameters (cf. LIST).

15.7.3 The command DELIVER

This command is used to specify the text which is to appear in the banner at the top and bottom of your line printer listings. The command takes a single parameter which should indicate where you want your output delivered. For example:

```
Command: DELIVER Chemistry K.B.
```

Note that the text can include any printable characters and spaces. If you wish to include a comma then the text must be enclosed in double quotes. For example:

```
Command: DELIVER "ABRO, Bush"
```

In order to find the current setting type:

```
Command: DELIVER =
```

The text specified will be printed on all listings produced after this time. It will not affect any listings which are already in the printer queue.

15.8 Interrogating the Device Queue

Section 15.6.1 describes the command DOCUMENTS which provides a progress report for all types of queued documents.

If you wish to find out whether your files have been listed you should use the command OUTPUT if you dispatched the file by using LIST or SEND.

15.8.1 The command OUTPUT

This is the command to use if you wish to find out about files you have sent to local devices or those accessible directly from the host by using the LIST or SEND commands, or output from programs which have been directed to these devices. The command takes two optional parameters. If you type OUTPUT with no parameter you will get a list of all such files which are still waiting in queues. Each one has an identifier.

If you wish to find out about all your files queued for a particular device then use the device name as the first parameter, for example:

```
Command: OUTPUT .LP
```

If you want more details about a particular file then type its identifier as the first parameter to OUTPUT, for example:

```
Command: OUTPUT 518
```

Note that there is a general command DOCUMENTS which provides a summary of all files awaiting action. See Section 15.6.1.

The OUTPUT command takes an optional second parameter to specify the destination of its output if you do not want it to be displayed on the interactive terminal.

15.9 Deleting Queued Documents

There are two commands for deleting files which are waiting to be printed on an output device. DELETEDOC should be used for documents sent by using LIST or SEND, or output directly from programs.

15.9.1 The command DELETEDOC

This command is used to delete any type of queued document:

* Files awaiting printing or plotting on an output device;

* Batch Jobs waiting to be run;

* Files awaiting transfer to another computer system.

Before you can use it you will need to know the identifier of the document. This can be found from one of the commands OUTPUT, BATCH, or TRANSFERS. The command takes one parameter, the identifier of the document. For example:

```
Command: DELETEDOC 518
```

If the document is currently active, e.g. a file being listed or a job being executed, then the activity will be terminated immediately.

15.10 Transferring Files to and from other Computer Systems

It is possible to transfer files to and from many other university and other computer centres by using the TRANSFER command. This is fully described in User Note 34. If you are using the command for the first time, or have forgotten the details of its parameter list you should just type:

```
Command: TRANSFER
```

You will then be prompted for each parameter in turn. If you are unsure what to reply in any case you should just type '?' followed by <RETURN>. You will then be given more information about the parameter.

15.10.1 Information about File Transfers

Normally the MAIL system is used to inform you about the success or otherwise of your file transfer requests. Additionally, there is a command TRANSFERS which can be used to find out about file transfer requests. It will also tell you about transfers to you which have been initiated on other systems. The TRANSFERS command is described in User Note 34.

15.10.2 The command HOSTS

This is the command to use if you wish to find out information about any host machine - e.g. its existence, its contexts (logon, mail and file services offered) etc. The command takes one optional parameter. If you type

```
Command: HOSTS
```

with no parameter you will enter the data base, where you will be prompted for further details.

If you wish to find out about a specific host then type its name as the parameter, for example

```
Command: HOSTS EMAS
```

If you wish to find out about several hosts at one site, then a search-on-key facility enables you to give any string as the parameter, for example

```
Command: HOSTS EDIN
```

will search for all hosts whose description or name includes the string EDIN.

15.11 Remote devices

Special commands are required if you wish to interact with devices which are not directly accessible to the host machine. These (NLIST, NDOCUMENTS and NDELETEDOC) are described in the following three sections.

15.11.1 The command NLIST

This command provides similar functions to LIST for devices which are not directly accessible from the EMAS-3 service machine. Its four parameters are identical to those for LIST (see above). When it is used for a particular device queue for the first time it will ask you to supply the name of a Host and a Username on that Host. The Host must be one which has a direct connection to the device involved. See HELP DEVICES for details of which devices are accessible from which hosts. This

additional information is needed because the operation is performed by transferring the file to the remote Host and then listing it as if from the specified Username.

After you have specified a Host and Username once it will normally be used every time you access that device by using NLIST. In the unlikely event that you wish to change the host and or username for a queue you should type:

```
Command: NLIST ?
```

You will then be prompted for all the parameters, including Host and Username.

15.11.2 The command NDOCUMENTS

This is the command to use to find out about the files you have listed by using NLIST. It takes one obligatory parameter, the queue (device) name for which you require information. For example:

```
Command: NDOCUMENTS DP23
```

Like NLIST it will need to know the name of the remote Host and a Username on that Host. It will prompt for this information on the first occasion you use it for a particular device. Additionally, each time you use it, it will prompt you for a password. You must respond with the Transfer password for the Username you are using on the remote Host. The output from this command will be returned in a MAIL message after a short delay. Meanwhile you can continue to use other commands. Note that this command can only be used if the remote host to which this device is attached is running on another EMAS system.

15.11.3 The command NDELETEDOC

This is used to delete files which have been dispatched to output devices by using the NLIST command. Like DELETEDOC it takes the identifier of the document as returned by NDOCUMENTS as its first parameter. When you first use the command it will ask for the name of the Host and the Username you used on that Host when you used NLIST. On subsequent occasions it will assume the same Host and Username. If you need to change these you should type:

```
Command: NDELETEDOC ?
```

and you will be prompted for all the parameters.

Every time you use this command it will prompt for a password. You should reply with the Transfer Password for the Username on the Remote Host. Note that this command can only be used if the remote host to which this device is attached is running on another EMAS system.

15.12 The command MAIL

This command enables you to communicate with other users on the host or on many other computer systems, both in Edinburgh and elsewhere. Using MAIL you can:

* Create and edit messages, which can consist of a few words or many pages.

* Send the messages to individual users, or to a list of users.

* Receive and read messages from other users.

* Reply to such messages, or Forward them to other users.

* File messages and locate them according to subject matter, or name of sender or recipient.

* List messages on output devices.

The MAIL command also gives you access to a number of Bulletin Boards. These contain a variety of notices, announcements, reports etc. on topics relating to computing services.

The MAIL command is described in a booklet available from the Advisory service. Alternatively you can find out about it by typing:

```
Command: MAIL
```

and then responding HELP to the Mail: prompt.

CHAPTER 16
RUNNING WORK IN BACKGROUND MODE

16.1 INTRODUCTION

The usual way to access EMAS is via an interactive terminal. This is known as using the system in 'foreground mode'. For work which does not require interaction from the terminal you may find it more convenient to put the work in the Batch queue to be executed later in 'background mode'. This is particularly relevant for jobs which need large quantities of computer time or access to private magnetic tapes (see Chapter 21).

This Chapter explains how to create a job file, and how to submit it to the Batch job queue by using the DETACH command. It also describes the BATCH command which you can use to check the progress of your batch jobs.

16.2 Creating a Job File

You should create a CHARACTER file by using EDIT or ECCE (see Chapter 5) or some other text editor. It should contain the commands to be executed typed as you would type them on the interactive terminal. There is one restriction concerned with command parameters. They must be typed immediately after the command, with or without keywords. The mechanism whereby the Subsystem prompts for missing parameters is not available in background mode. If your program would normally read data from the terminal then this should be included in the file at the point at which you would type it.

16.3 The command DETACH

DETACH jobfile,time,log,name,priority,order,rerun,

after,unique,delivery,tapes,decks,maxlog

This command is used to put a job into the batch queue, for subsequent execution without an interactive terminal. The parameters are described in Table 16.1.

This command takes 13 parameters, of which you are likely to need only a few for any one call. It is, therefore, a command with which you can usefully exploit the keyword mechanism. For example if you wish to specify that a job is not to be run before 6 p.m. you could type:

```
Command: DETACH JOBFILE,,,,,,,18.00.00
```

It would be easier, and less prone to error to type:

```
Command: DETACH JOBFILE,AFTER=18.00.00
```

JOBFILE	The name of a CHARACTER file which contains the commands and data which define the job. Note that the format of commands should be exactly as you type them on the interactive terminal. The DETACH command makes a copy of the job file so you can alter or destroy the file immediately after detaching it without affecting the copy in the job queue.

TIME The cputime limit for the job, in seconds. The lower the limit the sooner your job will be executed. If your job needs more time it will fail "Time Exceeded" Currently, the default is 30 seconds.

LOG Use this parameter to say where you want the job journal to go:

.LPnn	for output on the printer nn
filename	will put output in a file (overwriting existing contents if any)
.NULL	will suppress the job journal

Default is .LP. This is also used if file creation fails for any reason.

NAME You can use this parameter to give the job a name to help you identify it if you use the BATCH command (see below) to follow its progress. Default is the name of the DETACHed file.

PRIORITY If PRIORITY=LOW then run this job after all normal priority jobs, but only charge half normal rate.

ORDER Use this parameter if you wish to ensure that a number of jobs are executed in a particular order. Normally jobs are selected for execution in an order determined by their times of submission and their requirements for time and tapes. You can ensure that jobs are run in a particular order, regardless of their individual requirements, by DETACHing them in order using an increasing value for ORDER. For example if you DETACH two jobs with ORDER=1 and ORDER=2 they will be run in order even though the second may require less resources than the first.

As a further sophistication you can operate this ordering for more than one set of jobs. If you use the form ORDER=n/m you can indicate that this is job n within set m. For example you could detach jobs by using ORDER=1/5 and ORDER=2/5. This will ensure that they are run in order within set 5, but will not delay execution of jobs in other sets.

RERUN Use this parameter if you wish to specify that this job is not to be re-run if the system crashes whilst it is running (i.e. RERUN=NO). Default is YES.

AFTER Use this parameter if you want to delay execution of this job until after a specific date or time. Any of the following forms are valid:

```
date time
time date
date
time
```

where date is in the format dd/mm/yy and time is in the format hh.mm.ss Note that if only date is specified then time is assumed to be 00.00.00, so AFTER=03/07/86 means any time on, or after 3rd July 1986. Use of this parameter does not guarantee that the job will be run at the time specified, it merely ensures that it will not be run earlier.

UNIQUE	Use UNIQUE=YES if you wish to ensure that this is the only job with this NAME in the queue of batch jobs for you. Remember that if you do not specify a NAME then JOBFILE is used as NAME.
DELIVERY	Use this parameter to specify delivery information on the job journal if it is to differ from the text currently set by the DELIVER command.
TAPES	Use this to specify the serial numbers of private magnetic tapes to be used by this job. If more than one use & to separate them. If write permit ring required follow serial number with '*'. Examples: JS7748 HG6678* AN7689&AB455*&AN1088
DECKS	Use this parameter to specify the maximum number of tape decks needed at one time by your job. If omitted it will default to the number of tapes you have specified by using the TAPES parameter.
MAXLOG	The maximum size (in Kbyte) that you want the job journal to reach. If this limit is exceeded the job will fail "Output Exceeded".

Table 16.1 DETACH parameters

Examples of the use of DETACH

```
Command: DETACH MYJOB,30                          (detach job with time limit of 30 sec.)

Command: DETACH TAPEJOB,TAPES=AB1233&AB1288*      (two tapes)

Command: DETACH PRINTJOB,LOG=.LP23                (output to .LP23)
```

16.4 Checking the progress of your Batch Jobs

If the DETACH command works successfully then you will get a message such as:

```
ERCC99:JOBFIL is Batch queue entry 414.  Time limit 30 secs.
```

When it has been executed you will get a message such as:

```
**ERCC99 02/07/86 15:27  Batch job JOBFIL completed
```

You can check on the progress of either one job or all your jobs by using the command BATCH. This can be used in the following ways:

* With no parameter. In this case you will get a single line report about each job you have detached which is running, or has still to be run.

* With a first parameter which is the Batch queue entry number, returned by the DETACH command (see above). e.g.

```
Command: BATCH 414
```

In this case you will get further details about a single job. If you forget the queue entry number then you should use BATCH with no parameter. You should be able to identify the job you are interested in from the output of this command.

The command BATCH takes an optional second parameter to specify the destination of its output. Normally it is sent to the interactive terminal. For example:

```
Command: BATCH ,TEMP
```

16.5 Deleting a job in the Batch Queue

If you decide you do not want a queued job to be run you should use the command DELETEDOC. It takes one parameter, the Batch queue entry number, which is returned by DETACH, or which can be found by using the BATCH command. For example:

```
Command: DELETEDOC 414
```

16.6 Setting up initial conditions in a Batch job

The OPTION command, (see Section 17.9) includes a keyword BSTARTFILE. Using this option you can nominate a file of commands to be executed automatically at the beginning of each batch job.

CHAPTER 17
ADVANCED USE OF COMMAND LANGUAGE

17.1 INTRODUCTION

Chapter 2 introduces the EMAS-3 command language and describes how to type commands and their parameters. This chapter provides more information about the command language, including details of the wild-card facility which allows you to carry out an operation on a whole set of files.

This chapter also describes other mechanisms to make it more convenient to use EMAS-3 commands.

* The command history mechanism allows you to reissue commands, optionally with corrections, with very little typing.

* The OBEY command allows you to obey a sequence of commands repeatedly.

* The Macro Scheme allows you to create complex commands from a number of simple commands.

* The ALIAS command allows you to use abbreviated names for commands you use regularly.

This chapter also includes a description of the commands CPULIMIT and METER.

Finally there is a description of the OPTION command. This allows you to specify and interrogate options relating to various aspects of the Subsystem. Some of these are described more fully elsewhere in this Guide.

17.2 Calling Commands

The following examples illustrate the various ways in which a command can be called at the terminal.

1.
```
Command: analyse myfile
```

   This shows the straightforward case, where the required parameters are specified on the same line as the command.

2.
```
Command: analyse
File: myfile
```

   Here, EMAS prompts for the required parameter `File:` and supplies defaults for the other parameters.

3.
```
Command: analyse ?
File: myfile
Options():
Out():
```

   By specifying "?", the user requests that every parameter should be prompted for. Where the parameter is optional, its prompt indicates a default value in parentheses. Pressing <RETURN> causes that value to be used.

a. `Out(): ?`
   `<help text about the Output parameter>`
   `Out():`

   Typing "?" in response to a prompt causes help information about the parameter to be displayed.

b. `Option(): <CTRL+Y>`

   A response of `<CTRL+Y>` to any prompt causes the call to be abandoned and a return to command level.

4. `Command: analyse ??`
   `<help text about the command>`
   `File:`

   This call causes help text for the whole command to be displayed, followed by prompts for each parameter.

5. `Command: analyse myfile , output=.lp`

   As an alternative to entering parameters in their fixed order, when prefixed `keyword=` they can be given in any order. This is useful for commands with long parameter lists (the standard command OPTION is usually called with keyword parameters for this reason, e.g. `OPTION RECALL=PERM`). After the first keyword parameter encountered, all the remaining parameters must also be specified by keyword. The keyword name is the same as the first word of the prompt, but may be typed in an abbreviated form. For example `ou=` is equivalent to `output=`.

The following examples show ways in which multiple calls of a command can be specified in a single command line.

6. `Command: analyse teach.*`

   Where a wild-card expression is given, the set of matching files is found and the command is called in turn for each file in the set. In this example, all the files and groups in group `teach` are analysed (the asterisk matches any sequence of zero or more characters).

7. `Command: rename tx* , text*`

   For commands with two or more filename parameters, a wild-card expression for the first parameter may require a corresponding expression for the others. In this example, all files and groups beginning with `tx` are renamed to begin `text`.

8. `Command: insert br1y & br2y & aby , mydirectory`

   Another way of generating multiple calls is to use ampersanding. The above call causes three files to be inserted in `mydirectory`.

9. `Command: insert br*y & aby , mydirectory`

   Ampersanding and wild-carding may be combined. This call is similar in effect to the example before it.

10. `Command: analyse @flist`

    In this example, the character file `flist` is read and each line in turn is used as the parameter list of a separate call of the command.

17.3 The Command History Mechanism

The Command History Mechanism allows you to reissue previous commands (with corrections if required) with a minimum of typing. In its simplest form it works as follows:

```
Command: METER

06/07/86 16.24.46 CPU=0.75 Secs CT=4 Mins PT= 269 Ch= 17p

Command: !

Command: METER

06/07/86 16.25.25 CPU=0.76 Secs CT=5 Mins PT= 277 Ch= 18p
```

The single exclamation mark typed in response to the `Command:` prompt causes the last command line, including parameters if any, to be echoed and then executed.

17.3.1 Reissuing earlier commands

Command lines are preserved from one session to another. Earlier commands can be reissued by typing enough of their name after the exclamation mark to identify them:

```
Command: !IM

Command: IMP SOURCE27,Y27,.LP15
```

Alternatively you can type e.g.

```
Command: !-1
```

to execute the command line before the most recent one (etc.). The form:

```
Command: !ABC!
```

causes execution of the last command line containing `ABC`.

17.3.2 Displaying stored command lines

If you type `!=` in response to the `Command:` prompt the Subsystem will display all the stored command lines with an associated key number against each. As an alternative way of identifying a command to reissue you can type its key number:

```
Command: !3

Command: FORTRAN FSRCE.ROUT22,ROUT22Y,ROUT22L
```

17.3.3 Repeating commands with corrections

If you mis-type a long command line you may find it convenient to use the history mechanism to correct it. For example:

```
Command: EDIT SUBSYS.EMAS3SPECS,MYSPECS
```

This would fail because the full stop should be a colon. So now type:

```
Command: !.!:!
```

and the command will be reissued correctly:

```
Command: EDIT SUBSYS:EMAS3SPECS,MYSPECS
```

This has the effect of repeating the most recent command line containing '.', replacing each occurrence of '.' with ':'.

Note that the rightmost exclamation mark can be omitted provided that the replacement string does not end in comma and is not intended to be null.

To re-use the parameters used in the last command type e.g.

```
Command: NEWCOMMAND!
```

Conversely, to use the same command again but with new parameters type:

```
Command: !  NEWPARAMS
```

17.3.4 Other features of the command history mechanism

The above forms are the most commonly useful forms of repetition with alteration. The remainder are given for completeness.

`NEWCOMMAND!3`	perform `NEWCOMMAND` with same parameters as on command line having key number 3.
`!3 NEWPARAMS`	repeat command of command line number 3 with new parameters.
`NEWCOMMAND!ABD`	perform `NEWCOMMAND` with the same parameters as on most recently-stored command line commencing `ABD`.
`!ABD NEWPARAMS`	repeat most recent command line commencing `ABD` with new parameters.
`NEWCOMMAND!ABD!`	perform `NEWCOMMAND` with the same parameters as on the most recently-stored command line containing `ABD`.
`!ABD! NEWPARAMS`	repeat most recent command line containing `ABD` with new parameters.
`NEWCOMMAND!ABD!DE!`	take most recent command line containing `ABD`, replace each occurrence of `ABD` with `DE`, and perform `NEWCOMMAND` with the parameters from the resulting command line.

!ABD!DE! NEWPARAMS	take most recent command line containing ABD, replace each occurrence of ABD with DE, and perform the command of the resulting line with new parameters.

17.4 The OBEY command

This command provides a simple mechanism for executing a series of commands, and in-line data if needed, repeatedly. To use the command you first need to create a CHARACTER file by using a text editor such as EDIT or ECCE. The file should contain a sequence of commands and data typed as you would type them on your interactive terminal. Note the following points:

* Each command in the file must have its parameters following it on the same line (using continuation lines if required). You cannot use the mechanism whereby the command prompts for missing parameters.

* The data should follow the command in the same sequence and format that you would use if you were running the program from your interactive terminal.

To use the command type e.g.

```
Command: OBEY COMFILE
```

where COMFILE contains the commands and data.

OBEY takes an optional second parameter to specify a destination for its output. This can be the name of a file or suitable output device: the default is .OUT. For example:

```
Command: OBEY COMFILE,.LP15
```

Notes

* You can control the extent to which OBEY displays its progress on the terminal by using the OPTION command - see Section 17.9.

* The DETACH command (Section 16.3) uses a command file with the same format as OBEY. It puts the file into the Batch job queue to be executed later without an interactive terminal.

* OPTIONs FSTARTFILE and BSTARTFILE (see Section 17.9) allow you to specify the names of files to be OBEYed automatically at the start of foreground sessions and batch jobs respectively.

17.5 The Command Macro Scheme

A more sophisticated system of creating files of commands and data for execution as a single unit is provided by the Macro scheme. This effectively allows you to create your own commands without having to use a programming language. It is fully described in User Note 81.

17.6 Abbreviating Command Names

Most EMAS-3 command names are, or are closely related to, English words. This is in contrast to some other systems which use cryptic abbreviations for their commands. The EMAS naming policy does have the disadvantage that some command names are rather long. If you wish to you can abbreviate the names of commands you use frequently, right down to a single letter. Such abbreviations are not cryptic to you because you can choose your own abbreviations. The method is to use the command ALIAS. This normally takes two parameters. For example if you wish to use the letter 'F' for FORTRAN you should type:

```
Command: ALIAS FORTRAN,F
```

From now on you can use F as an alias for FORTRAN. Should you ever wish to remove the alias for a command then you should omit the second parameter:

```
Command: ALIAS FORTRAN
```

This would have the effect of removing all the aliases you have provided for FORTRAN.

Note that the aliases are stored in your Active Directory. For most of the time this is the file :SS#DIR. If you use ALIAS when you have selected another Active Directory then the information will be stored in it. (See Section 7.6.2 for an explanation of the Active Directory.) If you wish to find out what aliases you have set use the command:

```
Command: ANALYSE :SS£DIR,A
```

17.7 The command CPULIMIT

This command can be used to limit the amount of cputime which is used by a command. The default limit is very large, so it will not be necessary to set this in order to run a long program. Rather, it is used to prevent a program using too much computer time. If, for example, you have a program which, under some circumstances, appears to loop indefinitely you should use this command before you run it. The command takes two parameters, minutes and seconds, of which at least one must be set.

```
Command: CPULIMIT 5          5 minutes

Command: CPULIMIT ,30        30 seconds

Command: CPULIMIT =          displays the current setting.
```

17.8 The command METER

The METER command is used to display usage information and an indication of the cost of the current session so far. The output is of the form:

```
06/07/86 23.12.34  CPU= 10.28 Secs CT= 168 Mins PT= 3333 Ch= 228p
```

The information displayed is:

* Current date and time

* CPU time used, in seconds

* Interactive terminal connect time, in minutes

* Number of page turns

* Approximate charge at standard rate. This does not take account of the surcharge for commercial users.

17.9 The command OPTION

This command is used to change or interrogate a number of characteristics of the Subsystem for your use of EMAS. The command can be used in three different ways:

```
Command: OPTION keyword=value
```

This is the way to change one setting. The keywords and acceptable values are tabulated below.

```
Command: OPTION ?
```

In this case each keyword will be printed together with its present value. If you wish to retain the present setting then just press <RETURN>. Otherwise type the new value and then press <RETURN>.

```
Command: OPTION =
```

This will display all the keywords and their present settings.

ARRAYDIAG	With this option keyword, you can specify the number of elements of arrays to be printed in diagnostics. Example: `Command: OPTION ARRAYDIAG=25`
BLANKLINES	This can have a value of ON or OFF to indicate whether you want totally blank lines to be printed on your interactive terminal. By setting it OFF you can increase the amount of information displayed on your screen, at the cost of some clarity.
BSTARTFILE	This option is used to specify a file which is to be OBEYed at the start of every background (batch) job. If you want to stop using such a file then set this parameter to the value ""
CONTROLCHARINTS	This controls the interpretation by the EMAS front end processor of the control characters <CTRL+A>, <CTRL+C>, <CTRL+T> and <CTRL+K>. With CONTROLCHARINTS=ON, these control characters are treated as though the corresponding Int: character had been generated at the terminal. With CONTROLCHARINTS=OFF, these characters are passed to the mainframe unchanged (see Section 14.6.2).
DEFAULTDEFINESIZE	This value is used by the DEFINE command and specifies the maximum size (in Kbyte) of the output file which is being DEFINEd (see Section 13.6.3).

DEFAULTTERMINALTYPE The Subsystem "remembers" the most recent TERMINALTYPE settings and associates with them a corresponding network address. However, some network addresses are subject to variation depending on call routeing, and hence in some locations the same terminal address will rarely apply for a given terminal connection. If the Subsystem is unable to make a match on terminal address at log-on, and hence is unable to guess the user's terminal-type, the value of this parameter is taken as the TERMINALTYPE setting (see Section 14.4).

ECHO — This controls the extent to which the contents of an OBEY file are echoed when it is executed (see Section 17.4). Choose from:

FULL - to echo every line in the OBEY file

PART - to echo only the commands in the OBEY file

NO - if you don't want any of the OBEY file to be echoed.

FSTARTFILE — Use this to specify a file to be OBEYed automatically at the start of each interactive session. If you want to stop using such a file use the value "" for this parameter.

INITWORKSIZE — Use this to set the initial size of the Subsystem work file (in Kbyte). The default will normally be suitable.

ITINSIZE — Use this to specify the size (in Kbyte) for the input buffer for your interactive terminal. Normally the default is suitable. You may have to increase this if you transfer files directly from a disc on a microcomputer into your EMAS process.

ITOUTSIZE — Use this to specify the size (in Kbyte) for the output buffer for your interactive terminal. Normally the default value will be suitable.

PARITIES — This parameter controls the stripping of the parity bit (value 128) from bytes input at the terminal. When `PARITIES=OFF`, bit value 128 is removed from input bytes. When `PARITIES=ON`, input bytes are passed to users' programs unchanged.

PRELOADFILE — This can be used to nominate a file which is to be loaded at the start of each session and batch job. Having been loaded the file would remain loaded for the rest of the session. If you wish to stop using such a file you should set this parameter to the value ""

RECALL — This option is used to control the recording of your interactive terminal input and output for subsequent examination by using the RECALL or RECAP commands (see Section 14.7).

	Use one of the following values for RECALL: PERM – if you wish to store the information from one session to another. TEMP – if you only wish to store the information for the current session NONE – if you do not wish to use the facility.
USTACKSIZE	This is used to specify the maximum size (in Kbyte) to which the user program stack will be allowed to extend. Most programs will run in a stack of default size. This value will have to be increased for programs with very large numbers of variables.

Table 17.1 OPTION keywords

CHAPTER 18
HOW TO WRITE YOUR OWN COMMANDS

18.1 Introduction

On EMAS it is possible to write conventional programs which are executed by using the RUN command (Section 7.4.1). In this case the program is identified by its filename. As an alternative, it is possible to write programs to be executed as commands, just like other EMAS commands. There are two significant advantages when the programs are used:

* It is possible to choose a name for the program which describes its function.

* It is convenient for programs which require small quantities of data to be typed at the terminal.

For example a program could be written to compare two files and count the number of lines that differed. If it was written as a conventional program then the dialogue when it was executed might start:

```
Command:RUN CONLIB:TEXT.DIFFLINEY
Filename1:MYPROCS1
Filename2:MYPROCS2
```

Although this would work it is much more convenient to write the file to be called as a command, in which case the dialogue when it was executed would start:

```
Command:DIFFLINES MYPROCS1,MYPROCS2
```

This chapter explains how you can write your programs to be called as commands. Most of the examples are given in IMP, but all the routines described are callable from Fortran, and it is intended that they should be accessible from all other programming languages mounted on EMAS-3.

An alternative way to create your own commands is to use the Command Macro Scheme which is described in User Note 81.

18.2 Command structure

A simple command which reads a file of input and generates a file of output is given:

```
%externalroutine make %alias "c#make"
   %externalroutinespec emas3 integer(%stringname vector, %integername value)
   %externalroutinespec emas3 set return code(%integername flag)
   %integer inchan, outchan, flag
   emas3integer("Input;route,in,char;?;The file of directives", inchan)
   emas3integer("Output;route,out;.out;?;Where to output the report", outchan)
   select input(inchan)
   select output(outchan)
      {the command does its work}
   emas3 set return code(flag)
%end
```

The points to note are:

- the entry point name begins C# to distinguish commands from other routines (although when called at the terminal, the C# prefix is never included). This is part of the general naming scheme in use for all EMAS-3 entry names, intended to reduce problems of name conflicts. See Section 18.11 for details of how to set this form of name in commands written in languages other than IMP.

- the parameters are obtained by using (in this case) the routine EMAS3 INTEGER rather than by a `%string(255)` parameter to MAKE.

- The command does not need to perform any checking on the parameters.

- a return code is set before exit from the routine. This does two things:

  * allows job control statements to check the success or failure of command calls, and act accordingly;

  * prints an appropriate error message if the flag is positive.

  See Section 18.9.

- although this example is written in IMP, the same framework is appropriate for commands written in other languages, such as Fortran.

18.3 The Parameter Acquisition Mechanism - PAM

A command gains the benefits of the features described in the section on typing commands (17.2) by making use of PAM. There are three PAM routines, used to acquire parameters of types %string, %integer and %longreal. In calling a PAM routine, the command describes the parameter it requires and nominates a variable to receive the given value. A call is made for each parameter in turn, the order of call determining the order in which they are expected from the user.

No failure flag is returned to the command by any of the PAM routines. If a parameter is faulty, i.e. does not satisfy the qualifier in the vector supplied to the PAM routine, and the parameters are being acquired *by prompting,* the user will receive an error message and be prompted for a correct value. If the parameters are supplied on the command line, or if the command is running in background mode, then a failure message is printed and an immediate return to command level is made. This is all controlled by PAM; the command writer does not need to include any code to handle this.

The three PAM routines share a standard format:

```
(PAM vector, local variable)
```

In a simple case, a Fortran command which expects two parameters might make PAM calls:

```
call emas3string('Input file', cfile)
call emas3string('Operation', ops)
```

which simply nominate a prompt string and local variable name for each.
On executing each of these calls, PAM extracts the next parameter from the command line (or prompts the user if it is not in the command line) and then checks the validity of the value given.

In the example shown above, the PAM vector contains only a prompt field. Normally, it will contain additional fields to enable PAM to give help information, and to perform checking on the parameter. A complete PAM vector normally contains four *fields*, separated by semicolons, and is encoded as a text string:

```
prompt; qualifier; default; help
```

Examples of PAM vectors are given in the following section, and PAM fields are described in detail in Section 18.5.

18.4 Typical PAM calls

To anticipate the detailed description of PAM vectors given in the following section, here are examples of calls for the most commonly required parameters.

1. `emas3integer("Input;route,char,in;?;The file to be processed", inchan)`

   returns a channel number associated with the file specified by the user, which can be used in an IMP `select input` statement or a Fortran `READ` statement. The PAM vector calls for a file which must exist, and have filetype "character"; the name of the file is not returned, only the channel number used to read it. The "?" in the vector indicates that no default value is specified.

2. `emas3integer("Output;route,char,out;.out;". %c`
   `"The output file or device", outchan)`

   returns a channel number associated with a file or device, valid for output. The `char` tag is an extra check that if there is an existing file of the specified name with a filetype other than character, it will not be overwritten. The tag `data` can replace `char` where appropriate. The default value specified, `.out`, causes output to be directed to the terminal if no parameter is specified by the user.

3. `emas3string("File;fileormem,read,char;?;The file of records", fname)`

   returns the full name of a file, or member of a partitioned file, checked for `read` access permission and filetype "character". This contrasts with the previous examples where a channel number is returned.

4. `emas3string("Outfile;file,write;xreport;The report file", freport)`

   returns a filename (`xreport` by default) for which `write` access is required.

5. `emas3string("Operation;word,make,replace,append", ops)`

   returns a string containing one of the three words specified. In this example no default is defined. The user can specify an abbreviated form of the word, so long as no ambiguity results. Hence the value `m` would match `make`. Matching is insensitive to the case of the letters of the words in the vector and of the word typed by the user. The matched word is returned to the command in upper case.

6. `emas3integer("Operation;word,make,replace,append;make", opi)`

   returns an integer value, 0 for `make`, 1 for `replace` and 2 for `append` depending on which the user specifies. In this case `make` is nominated as the default if the user does not specify a value.

18.5 The PAM vector

As outlined in Section 18.3, the command writer describes the parameters the command requires by specifying a PAM vector for each. On each call of a PAM routine, the associated vector governs the behaviour of PAM in acquiring a value for the parameter. The PAM vector is encoded as a text string with up to five fields:

```
prompt; qualifier; default; help; keyword
```

The fields are described below.

18.5.1 Prompt field

The prompt field is displayed to the user when PAM requests input from the terminal for a parameter. This occurs when:

- the user fails to supply a value on the command line for a mandatory parameter (i.e. one with no default);
- the user explicitly requests prompting by calling the command with the single parameter "?" or "??".

The first word of the last line of the prompt is taken as the parameter's `keyword`, used where the form of call `keyword=value` is employed (see Section 17.2). If the prompt field contains a newline, then the first time it is issued the whole text is displayed, but subsequently only the last line is given.

18.5.2 Qualifier field

The qualifier field is particularly important. It indicates which values are acceptable for the parameter. This enables checks to be made on file and device names; on filetype, access permission and existence; or, for integer values, checks on specific values or ranges of values. The qualifier can be extended by appending tags to indicate additional checks. For example, the qualifier `file` may be made to check that the file exists by adding the appropriate tag: `file,exist`.

As an alternative to the set of built-in qualifiers, you can nominate your own checking routine to perform non-standard checks; this is described in Section 18.13.

18.5.3 Default field

If the command writer provides a value in the default field, then this is returned when the user does not offer a value. If the parameter is mandatory, then the default field should contain "?" to indicate that no default is available. When a user is prompted for a parameter and a default is available, it is shown in brackets:

```
Continue(yes):
```

Where the user omits a parameter and a default is available, then normally it is used directly without further reference to the user. However, it is occasionally desirable to have a parameter behave like a mandatory parameter while nevertheless possessing a default value. The effect is that a prompt is issued if no value is specified. The user can indicate acceptance of the default by pressing <RETURN> in response to the prompt. This conditional defaulting is selected by enclosing the default field in parentheses:

```
"Input file;file;(t#list);some help"
```

If a null value is acceptable for a parameter, the tag ornull should be given. If, in addition, the default value is itself null, this should be explicitly shown:

```
"Output;file,ornull;;some help"
```

In response to a prompt, a null value can be typed as "". This may occasionally be useful when a null value is acceptable for a parameter, but a non-null default is defined for it.

18.5.4 Help field

The help field is displayed when the user replies "?" in response to a prompt. Newlines may be included in the text of the field. An alternative to supplying the text of the help information explicitly, is to nominate a helpfile for the command. Indeed, this is the required method of providing help information about the whole command, when the user requests this by entering the single parameter "??" (see Section 17.2). A helpfile is a character file constructed as a VIEW file (see User Note 9). One helpfile may contain entries for several commands, with a "view section" for each. Within the section describing each command, there should be a subsection for each of its parameters. The help text for the ACCEPT command is as follows:

```
!<ACCEPT
   Command:ACCEPT file[,newname]

This command is used in conjunction with the OFFER command to
transfer a file from one user to another.  First the owner must OFFER
the file to you.  At any time after it has been OFFERed you can use
this command to ACCEPT it.  If the command succeeds the file will be
transferred into your file index.
!<FILE
The full name of a file which has been OFFERed to you by another user.
!>
!<NEWNAME
Use this parameter if you want to change the name of the ACCEPTed
file.  Otherwise the only part of the name that will change will
be the username.
!>
!>
```

To designate a helpfile, the text of the help field should appear in the following form within the vector:

```
call pamhelp(ercc90:myhelpfile)
```

In general, a helpfile need only be specified in the vector of the first PAM routine call made by a command. Its value is remembered for all subsequent PAM routine calls made for the command.

For a command with no parameters, no PAM routine call is made and therefore no help field is defined. The procedure EMAS3 HELPINFO provides an alternative method of conveying a help field to PAM. The help field given is used only if the user requests help about the command by calling it with the special parameter "??". The specification of the routine is:

```
%external %routine %spec EMAS3 HELPINFO(%string %name HELP FIELD)
```

It is also possible to nominate your own help routine to supply the necessary information: see Section 18.12.

18.5.5 Keyword field

As described in Section 18.5.1, the keyword of a parameter is taken from the first word (of the last line) of the prompt field. Exceptionally, where this value is unsuitable, an additional field can be appended to the PAM vector, following the help field and separated from it by a semicolon, to define the keyword. For example:

```
"File;fileormem;?;The type4 entries;type4"
```

18.5.6 Missing PAM vector fields

If a field is omitted from the PAM vector, a value is allocated as follows:

prompt	is set to	Integer (or String or Long Real, depending on the name of the PAM routine called)
qualifier	is set to	any
default	is set to	? (i.e. no default)
help	is set to	the qualifier field.

18.6 Getting a string parameter

The routine for obtaining the value of a string parameter is EMAS3 STRING. Its specification is:

```
%external %routine %spec EMAS3 STRING(%string %name VECTOR, VALUE)
```

The qualifier field in the vector is used to determine whether the parameter value supplied by the user is valid. The possible qualifiers are:

qualifier	valid values
any	any non-empty string
group	a syntactically correct group name
fileormem	a syntactically correct file or pdfile member name
file	a syntactically correct file name
fileordev	a file name or a network device name
user	the name of a registered user
device	a device name
filelist	one fileormem, or several joined by '+' signs
date	a date, in the format dd/mm/yy

`word,text0,text1,text2,...`
the parameter supplied is compared with text0, text1, etc. If an unambiguous match is found, ignoring case and spaces, the matching text in the qualifier is returned. For example, given the qualifier

```
word,no,yes
```

a parameter value of ye would yield the result YES. The result is returned in upper case. A null value is defined by making one of

the text options null, e.g. word,,yes,no.

lcword,text0,text1,text2,..
operates like word except that the result is returned in lower case.

18.6.1 Tags for string qualifiers

Qualifiers may be modified by tags. Tags follow the qualifier and are separated from it, and from each other, by commas.

String parameters are normally returned in upper case with spaces removed. This can be varied by use of the following tags:

verbatim	the result has case and spaces preserved
lc	the result is returned in lower case.

The secret tag is provided to hide confidential information, such as a password parameter retrieved by using the qualifier any.

secret	the effect of this tag varies according to how the parameter was acquired. If it was presented at Command level, then the value is removed from the recall file. If prompted for then echoing is turned off.

For files, pdfile members and groups, the existence of the object and access permissions to it are checked by the following tags:

exist	the file, member or group must exist
notexist	the file, member or group must not exist
read	read access must have been permitted (implies exist)
write	write access must be allowed (if the file exists).

The following tags apply a filetype check to a file or member, if the file or member exists. In themselves, they do not imply an exist check.

object	an object file
nonst	a non-standard file (not object, char, data or pdfile)
char	a character file
data	a data file
pdfile	a partitioned data file
dir	a directory file (associating entry names with object file names).

If none of these tags is set, the parameter is merely checked for correct syntax.

The join tag applies to the filelist qualifier only:

join	concatenates the component files to a temporary file whose name is returned. If only a single file is specified, no copy is made of it.

Only character files may be joined.

All parameters are subject to ampersand expansion (see Section 17.2). To forbid this, the following tag is used:

noamp	the parameter does not permit ampersand expansion.

All file, member and group parameters are subject to wild-card expansion (see the examples in Section 17.2). This can be controlled by the following tags:

nowild	the parameter does not allow wild-card expansion
cowild	if a wild-card expression is given for this parameter, then all other parameters with tag cowild require a corresponding wild-card expression.

Certain standard parameter values may be acceptable even though the value supplied does not pass the check implied by the qualifier, for example:

```
file,orequals
any,ornull
```

These parameters are catered for by the following tags:

ornull	a null value is acceptable
orequals	a value of "=" is acceptable
or.all	a value of .ALL is acceptable
or.end	a value of .END is acceptable
or.null	a value of .NULL is acceptable.

When parameters are supplied on the same line as the command, the command history scheme stores them along with the command name. By contrast, parameters acquired by prompting are not stored as part of command history. By including a tag history in the first PAM call, you can indicate that the parameters prompted for are to be added to the command history record.

18.7 Getting an integer parameter

If the user is required to supply an integer value for a parameter, then the command writer should use the PAM routine EMAS3 INTEGER to retrieve it. However, if the outcome of a PAM check on a string value is most naturally represented as an integer, then EMAS3 INTEGER should again be used. For example, a parameter might have three valid values, YES, NO, DONTKNOW. By using EMAS3 INTEGER and setting up the qualifier part of the vector appropriately, the command writer can make PAM check for these values and return 0, 1 or 2 respectively. See the qualifier word, below, for details.

Another situation where EMAS3 INTEGER is appropriate is in the setting up of a route (in EMAS 2900 terminology, defining a file or device). When the user supplies a file name or device name, EMAS3 INTEGER can check the validity of this parameter and return a channel number, which the command could then select for input or output. See the qualifier route, below, for details.

The specification for EMAS3 INTEGER is:

```
%external %routine %spec EMAS3 INTEGER(%string %name VECTOR,
          %integer %name VALUE)
```

The qualifier in the vector is used to determine whether the parameter value supplied by the user is valid. The possible qualifiers are as follows:

qualifier	valid value
any	any integer value
m:n	a range of numbers. m and n are the bounds; a null value means no lower (or upper) bound; for example 6:10 means a number between 6 and 10 3: means a number greater than or equal to 3 :3 means a number less than or equal to 3
a sequence of ranges or values in numerical order	for example 0:3,5,8,10:12 means 0,1,2,3,5,8,10,11 or 12 A null value is permitted by adding a tag of the form null=value. value is returned as the result if no parameter is supplied.
word,text0,text1,text2,...	the parameter supplied is compared with text0, text1, etc. If a text matches the parameter value or starts with the parameter value, ignoring letter case, then the index of the text is returned. For example, given the qualifier word,no,yes a parameter value of N or NO or n etc. would give 0, while y or ye or yes would give 1. A null value is defined by making one of the text options null, e.g. word,no,yes,.
route	a file or pdfile member name, or a device name.

18.7.1 Tags for the route qualifier

A number of tags are provided to modify the operation of the route qualifier. Tags follow the qualifier and are separated by commas. See the examples in Section 18.4.

As described above, the route qualifier allows the user to specify a file or device name. PAM associates this with a channel number which it returns to the command as an integer value. This may be used in a subsequent call of an IMP SELECT INPUT or SELECT OUTPUT or a Fortran READ or WRITE statement.

The normal procedure is to allow PAM to choose a free channel number. It is possible, however, to preset a fixed channel number in the VALUE parameter of EMAS3 INTEGER, and indicate this by use of the following tag:

preset	the file or device is associated with the channel number given in the VALUE parameter

Additional checks are performed when a tag is included which indicates whether the

route is to be used for input or output:

in	the file or device is to be used for input. Hence a check is made that either a valid input device, or an existing file with read access permitted, is specified.
out	the file or device is to be used for output. A check is made that either a valid output device, or a filename which can be written to, is specified. The file need not exist, in which case it must be possible to create it; if it does exist, however, it must have write access permitted.

If the file does exist, an additional check can be made on its filetype to prevent an attempt to read from a file of the wrong type, or to overwrite an existing file of a different type.

char	a character file
data	a data file
object	an object file
nonst	a non-standard file (mapped file).

Additional tags can be used to specify further details of the file definition. They have the same effect as the similarly named parameters of the DEFINE command.

size= kb	where kb is a number in the range 1 to maxfilesize, giving the maximum allowed size for the (output) file
Vn	defines V record format with a maximum of n bytes per record
Fn	defines F record format with n bytes per record
C	defines a character file (for Fortran only).

18.8 Getting a long real parameter

The routine for obtaining the value of a long real parameter is EMAS3 LONG REAL. Its specification is:

```
%external %routine %spec EMAS3 LONG REAL %c
        (%string %name VECTOR, %long %real %name VALUE)
```

The only qualifier provided is any.

18.9 Setting a Return Code

When a command has done its work it should set a return code before exiting. This is done by use of the routine EMAS3 SET RETURN CODE, which has the specification:

```
%external %routine %spec EMAS3 SET RETURN CODE(%integer %name FLAG)
```

If the command has failed, FLAG should be set to some non-zero value. If the failure is as a result of a Subsystem procedure failing (e.g. EMAS3 CONNECT or EMAS3 RENAME), then the return code should be set to the flag value returned by the Subsystem procedure. The effect of doing this is to cause an appropriate failure

message to be printed on the user's terminal, of the form

command fails *failure message*

An arbitrary text may be placed in *failure message* by setting FLAG=100 and setting the text required by using the following routine:

```
%external %routine %spec EMAS3 SET FNAME(%string %name TEXT)
```

18.10 Additional PAM routines

A number of routines are provided to enable a command to perform manipulation of the command line:

- `%external %routine EMAS3 GETRESTOFLINE(%string %name LINE)`

  returns the list of parameters remaining on the line, or the complete parameter line if none have been read yet. The routine is not destructive, and can be followed by calls of EMAS3 STRING or EMAS3 INTEGER as required.

- `%external %routine EMAS3 LINE(%string %name LINE)`

  again returns the remainder of the parameter line, but clears it in addition.

- `%external %routine EMAS3 LCLINE(%string %name LINE)`

  operates like EMAS3 LINE, but returns the parameter line in the same form as specified by the user. Hence spaces and case are preserved.

- `%external %routine EMAS3 SETLINE(%string %name LINE)`

  sets the value of the current parameter line.

- `%external %routine EMAS3 LASTPARAM(%string %name PARAM)`

  returns the most recently acquired parameter in the form of an uninterpreted string. If the previous PAM routine acquired an integer parameter with qualifier `route`, then EMAS3 LASTPARAM would return a string containing the file or device name specified by the user. If the previous routine called for a string with qualifier `filelist,join`, which returns the name of a temporary concatenated file, then EMAS3 LASTPARAM would return the original filelist parameter value.

The routines can be used to preview the parameters before formally acquiring them. The preview may be required to guide the subsequent parameter acquisition process. Alternatively, the parameter line may be amended, and additional parameters added, before EMAS3 SETLINE is used to put it back in place. Finally, the standard PAM routines are called to acquire each parameter in turn.

18.11 Commands written in other languages

The routines described in this chapter may be called by a command written in any language available on EMAS-3. The only amendment required is to convert the name of the procedure to the corresponding `C#name` form, so that it is explicitly recognized as a command. This is done by using MAKECOMMAND. The parameters to MAKECOMMAND are the name of the object file followed by zero or more procedure

entry names. A Fortran object file may contain one or more subroutines, but not a main program entry. If no procedure entry names are specified, then MAKECOMMAND converts all procedure entry names in the object file; otherwise MAKECOMMAND converts only the specified entry names.

```
Command: makecommand fortobj

Command: makecommand object2 , mylist , mycount , ftest
```

An example of a command written in FORTRAN which uses PAM follows:

```
      subroutine perman
      character*128 c
      character*130 d
      call emas3string('Filename;fileormem',c)
      d = c//',p'
      call emas3('analyse',d,iflag)
c     {other analysis of the file}
      call emas3setreturncode(iflag)
      end
```

18.12 User-provided help routines

A command writer can nominate his own routine to provide help information for a command and its parameters. The routine should be written with a specification:

```
%external %routine MYHELP(%string %name COMMAND, KEY)
```

The name of the routine is placed in the `help field` of the PAM vector (Section 18.5.4) as follows:

```
"File;fileormem;?;call myhelp"
```

If this routine is called by PAM (because the user asks for help), the first parameter gives the name of the command about which help is requested. The `C#` prefix is not included in the command name. If information on the whole command is required (i.e. the user typed "??"), then the second parameter is null; otherwise it contains the `keyword` of the relevant parameter.

18.13 User-provided checking routines

A command writer can choose not to use the standard qualifiers and tags in the qualifier field of the PAM vector, but supply his own routine to perform parameter validation. The name given to the routine is at the discretion of the user. This routine should have one of the following specifications, depending on the type of the parameter:

```
%external %routine MYSCHECK(%string %name COMMAND, KEY, INPUT
        %string %name OUTPUT VALUE, %integer %name FLAG)

%external %routine MYICHECK(%string %name COMMAND, KEY, INPUT
        %integer %name OUTPUT VALUE, %integer %name FLAG)

%external %routine MYLRCHECK(%string %name COMMAND, KEY, INPUT
        %longreal %name OUTPUT VALUE, %integer %name FLAG)
```

`command` gives the name of the command in which the parameter to be checked occurs.

`key` gives the name of the parameter to be checked.

`input` gives the text of the parameter typed by the user.

`output value`
receives the result of evaluating `input`. The type of this parameter must agree with the type of the PAM routine called - EMAS3 STRING, EMAS3 INTEGER or EMAS3 LONGREAL.

`flag` conveys an indication of the success or failure of the check. Zero indicates success. Any other value causes a failure message to be printed.

The name of the user-supplied checking routine is placed in the `qualifier field` of the PAM vector (see Section 18.5.2) as follows:

```
"File;call mycheck;?;The file of names"
```

CHAPTER 19
STORE MAPPING

19.1 INTRODUCTION

This chapter describes the facilities provided for accessing the contents of files by mapping them onto data structures in IMP programs and named COMMON blocks in Fortran programs.

19.2 PRINCIPLE OF OPERATION

As explained in Appendix 2, all files (except those held on magnetic tape) are accessed by connecting them at an address in a very large virtual memory. The languages available include routines such as READ to enable programs to access the contents of files in a conventional manner, and this provision expedites the transfer of programs to and from EMAS. On the other hand, these routines are necessarily inefficient, in that each character that is accessed has to be moved from one address in the virtual memory (within the file) to another address in the virtual memory (within the user program's data area). The facilities described in this chapter remove the requirement for this intermediate movement of data, and at the same time free you from any constraints imposed by file formats; for example, the restriction that IMP direct access files must have a block size of 1024 bytes.

If you program in IMP you will find information about the language facilities for store mapping described in the IMP 80 Language manual. If you program in Fortran you will need to understand how to use named COMMON blocks. This is described in the Fortran77 Language Manual.

19.3 FILES FOR DIRECT MAPPING

Any file type can be used for direct mapping, but three types are especially suitable:

* Character files: these have a very simple structure (see Section 13.3).

* Data files with fixed length records. The use of data files with variable length records is not recommended, since it would be easy to corrupt the file by altering the record length information which is interspersed with the data itself.

* Data files with no record structure: these files are intended for direct mapping. They have no pre-defined structure, and can only be used for this type of file access.

Note that a file to be used for direct mapping must already exist - it will not be created by the Subsystem merely as a result of being DEFINEd and then referenced by a program (see below).

19.3.1 Creating a store map file - the command NEWSMFILE

```
NEWSMFILE file, size
```

The command NEWSMFILE is used to create a store map file (i.e. an unstructured data file). It takes two parameters, both obligatory:

`file` the name of the file to be created.

size the length of user data required (in bytes).

For example, if you require a file to hold an array of 1000 variables, each of 8 bytes, you could use the command

```
Command: NEWSMFILE STORE,8000
```

A file created by this command has every byte in its data area set to zero.

19.4 USING STORE MAPPING IN IMP

19.4.1 Linking the file to a data structure within an IMP program

The DEFINE command is used to link a particular file to a channel number for use in the program. In this case only the first two parameters are relevant. For the above file one could specify

```
Command: DEFINE 27,STORE
```

Note that .NULL must not be specified in the DEFINE command when direct mapping is to be used.

Within the program it is necessary to include a call on the routine OPENSM to connect the file and determine the address in the virtual memory at which it is connected. The routine should be specified thus:

```
%EXTERNALROUTINESPEC OPENSM(%INTEGER CHAN, MODE, %INTEGERNAME AD, LEN)
```

CHAN is used to specify the channel number used for the DEFINE call for the file being used.

MODE can take one of the following values:

0 connect file in any mode
1 connect file for reading only
2 connect file for reading and writing.

AD returns the address of the start of data or zero if the file cannot be connected in the requested mode.

LEN returns the length of the data in the file in bytes, if the connection was successful.

Notes

* When you do not want to alter the contents of a file you should specify read mode.

* When you want to write to a file belonging to another user then this routine allows you to check that you have write access permission, and that no other user has the file connected in a conflicting mode.

* The routine allows you to deal in a controlled way with a failure to connect a file.

If the variables in the file STORE (above) are of type longreal then they could be accessed by mapping them onto an array. The method is to specify an array format and an array name, and to equivalence the array name to the file:

```
%INTEGER START, LEN
%LONGREALARRAYFORMAT OUTAF(1:1000)
%LONGREALARRAYNAME OUTA
%EXTERNALROUTINESPEC OPENSM(%INTEGER CHAN,MODE, %INTEGERNAME AD,LEN)
      :
      :
   OPENSM(27,2,START,LEN)
      :
      :
   OUTA==ARRAY(START,OUTAF)
```

From this point on, the contents of the file can be accessed as `OUTA(1)` to `OUTA(1000)`.

Note that OPENSM must not be called more than once in respect of a particular file, unless the file in question has been closed in the meantime (see below). Thus the information obtained from OPENSM - start address and length of user data - must be saved if required later in the program.

19.4.2 Effect of accessing the array

It should be understood that, in the above example, the array `OUTA` and the file onto which it has been mapped are one and the same. Thus if the following code were executed the effect would be to set to 0 the first 20 elements of the array - that is, the first 20 values in the file:

```
OUTA(I)=0 %FOR I=1,1,20
```

This change lasts not merely until the end of the program - it is permanently recorded in the file, and any subsequent access to this file will find these elements cleared to zero. Thus, where a file is modified as the result of a program run, it is clearly desirable to use direct mapping, since the file is thereby kept up to date at all times. This is particularly useful when the program terminates unexpectedly.

It is possible to use much more complex data structures if records are used, and the combination of record manipulation and direct file mapping provides a useful tool for applications requiring random access to large or complex directories or tables.

19.4.3 Mapping onto members of partitioned files

It is possible to map onto a member of a partitioned file. In order to avoid the possibility of corrupting other members it is only possible to use read access. Thus this mechanism is only of use for programs which use mapped files to read, rather than alter, the data.

19.4.4 Closing mapped files

At times it is necessary to close a mapped file. In this context closing implies removing the link between the file and the logical channel in the program. The result is to free the file for other use; for example, it might be necessary to access the file by another access method in the same program. The program must be written in such a way that after closing the file it makes no reference to the program variables that have been equivalenced to it. The effect of doing so is undefined but could include corruption of files other than the mapped one. This is because the area of virtual memory used for the mapped file might have been re-used for another file.

The routine used for closing a file must be specified:

```
%EXTERNALROUTINESPEC CLOSESM(%INTEGER CHAN)
```

and the call would be, for example:

```
CLOSESM(27)
```

19.4.5 Examples of store mapping using IMP

In the first example a record array name is mapped onto an unstructured data file. Some of the record elements are accessed.

```
%ROUTINE PAYCHECK(%INTEGER CHANNEL, RECNO)
%INTEGER I, J, K, START, LENGTH
%STRING(11) NAME
%EXTERNALROUTINESPEC OPENSM(%INTEGER CHANNEL, MODE, %INTEGERNAME A, L)
%RECORD %FORMAT PAYF (%STRING(11) SURNAME, %INTEGER AGE, SEX, YEAR,
                      %INTEGERARRAY SALARY (1:12))
! Each record thus formatted contains 72 bytes.
%RECORD (PAYF) %ARRAY %FORMAT PAYAF(1:RECNO)
%RECORD (PAYF) %ARRAY %NAME PAY
      :
      :
! Assume that a file was associated with channel CHANNEL via DEFINE,
! prior to the execution of the routine.

   OPENSM(CHANNEL,2,START,LENGTH); ! File now connected.
   %IF LENGTH < 72*RECNO %START
      PRINTSTRING("File too small:")
      WRITE(LENGTH,1); NEWLINE
      PRINTSTRING("Must be at least")
      WRITE(72*RECNO,1); NEWLINE
      %RETURN
   %FINISH
   PAY==ARRAY(START,PAYAF)
   ! Now %RECORD %ARRAY %NAME PAY has been mapped onto the file.
   ! Note that START was set by the OPENSM call.
      :
      :
   NAME=PAY(I)_SURNAME
   %IF PAY(I)_SALARY(J)>350 ....
      :
      :
   PAY(K)_YEAR=1978
      :
      :
%END; ! Of %ROUTINE PAYCHECK.
```

In the next example direct mapping is used to access the contents of a character file. The structure of a character file is described in Sections 4.2 and 13.3. This routine could be used to count the number of lines in the file - far more efficiently than by using READSYMBOL.

```
%EXTERNALROUTINESPEC OPENSM(%INTEGER A,B, %INTEGERNAME C,D)
%ROUTINE COUNT LINES(%INTEGER CHAN)
%INTEGER START, FLENGTH, I, LINES

   OPENSM(CHAN,1,START,FLENGTH); ! FLENGTH is set to number of bytes
                                 ! in the file.
                                 ! Channel CHAN must be DEFINEd
                                 ! before the program is executed.
   LINES=0
   %FOR I=START,1,START+FLENGTH-1 %CYCLE
      %IF BYTEINTEGER(I)=NL %THEN LINES=LINES+1
   %REPEAT
   PRINTSTRING("Number of lines in file = ")
   WRITE(LINES,1)
   NEWLINE
%END
```

In the next example the program prints out the numerical values of the bytes in a given record in a file which has fixed length records, and which has been DEFINEd as channel 80:

```
   %BEGIN
   %EXTERNALROUTINESPEC OPENSM(%INTEGER CHAN,MODE, %INTEGERNAME AD,LEN)
   %EXTERNALROUTINESPEC EMAS3PROMPT(%STRINGNAME S)
   %INTEGER RECL, RECNO, START, LEN, MAX, RECSTART
      OPENSM(80,1,START,LEN)
      EMAS3PROMPT("Record length:")
      READ(RECL); ! Length of each record.
      MAX=LEN//RECL; ! MAX is highest record no.
      EMAS3PROMPT("Record no:")
NEXT: %CYCLE
         READ(RECNO)
         %STOP %IF RECNO<1; ! End run.
         %EXIT %IF RECNO<=MAX
         PRINTSTRING("Beyond end of file")
         NEWLINE
      %REPEAT
      REC START=START+RECL*(RECNO-1); ! Start of required record.

      %BEGIN; ! New block to allow dynamic definition of BAF
      %BYTEINTEGERARRAYFORMAT BAF(1:RECL)
      %BYTEINTEGERARRAYNAME BA
      %INTEGER I
         BA==ARRAY(RECSTART,BAF)
         %FOR I=1,1,RECL %CYCLE
            WRITE(BA(I),3)
            NEWLINE %IF I&X'F'=0; ! Newline every 16 nos.
         %REPEAT
      %END

      ->NEXT
   %ENDOFPROGRAM
```

19.4.6 Changing the size of a mapped file

The routine CHANGESM can be used to change the size of a mapped file. It must be specified:

```
%EXTERNALROUTINESPEC CHANGESM(%INTEGER CHAN, NEWSIZE)
```

where

CHAN is the channel number on which the file is defined

NEWSIZE is the new size required, in bytes

Notes

* The file must be DEFINEd but must not be open; i.e. the call should be made *before* a call of OPENSM, or *after* a call of CLOSESM. In the latter case another call of OPENSM must be made after the call of CHANGESM and any mapping of an array onto the file must be repeated, because the file may have been moved to a different location in the virtual memory when its size was changed.

* NEWSIZE can be larger or smaller than the present size. If it is smaller, then any information beyond the end of its new size is lost. Otherwise the previous contents of the file remain unchanged. If NEWSIZE is larger than the present size, each additional byte is set to zero.

* It is recommended that the use of CHANGESM be restricted to unstructured data files, i.e. those that have been created explicitly for mapping by using the NEWSMFILE command.

Example:

In the following example the file STORE used in an earlier example (of NEWSMFILE) is extended to allow the program to access 1500 longreal variables; it is assumed that the command DEFINE 72,STORE has already been given. Notice that the program does not itself make explicit reference to STORE, only to the associated channel number.

```
%BEGIN
%EXTERNALROUTINESPEC CHANGESM(%INTEGER CHAN, NEWSIZE)
%EXTERNALROUTINESPEC OPENSM(%INTEGER CHAN, MODE, %INTEGERNAME AD, LEN)
%LONGREALARRAYFORMAT OUTAF(1:1500)
%LONGREALARRAYNAME OUTA
%INTEGER LEN, START
! File STORE is defined on channel 72.

   CHANGESM(72,12000); ! Extend to hold 1500 longreals (12000 bytes).
   OPENSM(72,2,START,LEN)
      :
      :
   OUTA==ARRAY(START,OUTAF)
      :
      :
```

19.5 USING STORE MAPPING IN FORTRAN

19.5.1 Outline of method

In contrast to IMP where the method of linking mapped files to programs is similar to that for other types of data files, Fortran mapped files are provided as a feature of the program loader. The entity which is mapped is a named COMMON block. Different COMMON blocks can be mapped to different files, or even to different areas within the same file.

Normally when an object file is loaded which contains a reference to a named COMMON block the loader searches for a file containing a matching DATA entry. This would be supplied by a BLOCK DATA subprogram file, which gives initial values for the variables in the COMMON BLOCK. If no such entry is found then the loader allocates space from the GLA (general linkage area) for the variables. The name, address and length of this area are recorded by the loader and any further files loaded which refer to a COMMON block of the same name are made to point to the same area.

When using store mapping you nominate a file which is to be used, instead of the GLA for the COMMON block. This must be done before the program is loaded by using the command DATASPACE. This command sets up a link between a DATA entry name and a file. After this has been done all references to the same DATA entry (i.e. all subprograms which contain a reference to the COMMON block involved) will have their pointers set to address the selected file. Thus all accesses to variables in the COMMON block will actually be references to the file, and any alterations made will be made to variables held in the file, and will be retained after the program has stopped running.

19.5.2 The command DATASPACE

This command is used to associate a COMMON block name with a file. It takes five parameters of which the first three are obligatory.

```
DATASPACE entry,file,length,offset,access
```

The parameters are used as follows:

`entry` is the name of the COMMON block to be mapped.

`file` is the name of the file which contains the area of store to be mapped.

`length` is the length of the COMMON block, in bytes.

`offset` (optional) is the offset from the start of the file of the area to be used. This parameter has a default of zero, meaning that the COMMON block is mapped on to the first `length` bytes of the file.

`access` (optional) this can be R or W for read or write. By default it is W for a file and R for a member of a partitioned file. If your program does not alter any of the values in the COMMON block then you should use R.

19.5.3 Example of Fortran store mapping

In the following fragment of a Fortran program reference is made to a COMMON block with the name `STAR`. This block contains 100 REAL variables followed by 101 INTEGER variables.

```
      COMMON /STAR/DIST(100),IBRIT(100),IRUN
C First update run number
      IRUN=IRUN+1
100   FORMAT(' Astro-Record Program - Run No:',I6)
      WRITE(6,100)IRUN
```

In the first few lines of code the variable IRUN is incremented and printed. This is used to indicate the number of times the program has been executed. Later code could read from or alter any of the elements in DIST and IBRIT. Any alterations made would, like the updated run number, remain in the file until the program was run again.

Before the program was run the first time you would need to create a file to hold the information. The command NEWSMFILE, described above, could be used:

```
Command: NEWSMFILE STARFILE,804
```

The value 804 allows space for 100 REALS (400 bytes) and 101 integers (404 bytes). Before running the program you would have to type the command:

```
Command: DATASPACE STAR,STARFILE,804
```

to tell the loader where to put the COMMON block STAR.

19.5.4 Removing the link between a file and a COMMON block

Once a link has been established by the use of DATASPACE it remains intact until you log off, or until you use the command:

```
Command: RESETLOADER
```

This command resets the loader tables to the state they were in when you first logged on, and thus any links between files and COMMON areas are lost. You will only need to use this mechanism if you want to carry out some operation on the file (such as OFFERing it to another user), or if you wish to link the COMMON block to a different file.

19.5.5 Using more than one mapped COMMON block

It is possible to use DATASPACE more than once to link different COMMON blocks and files. You may find it useful to map more than one COMMON block onto the same file. In order to do so you need to use the offset parameter for DATASPACE. By default DATASPACE aligns the COMMON block with the start of data in the file. If you specify an offset then the COMMON block will be offset by the number of bytes specified from the start of data in the file.

For example, using the program above you might wish to map another COMMON block onto the same file as you use for STAR. If the second COMMON block has a length of 200 bytes and is called OBSERV then you could make calls of DATASPACE such as:

```
Command: DATASPACE STAR,STFILE,804

Command: DATASPACE OBSERV,STFILE,200,804
```

Notes

* The block OBSERV has been offset by 804 bytes from the beginning of the file STFILE. In this case it follows immediately after STAR.

* COMMON blocks must not overlap - you could not have used an offset of 800 since this would have meant that STAR and OBSERV would have overlapped by 4 bytes.

* The file STFILE would have to be at least 1004 bytes long.

19.5.6 Choice of access mode

If your program alters the contents of the mapped COMMON block then it must connect the mapped file in write mode. This is the default for normal files so there is no need to use the access parameter in the call of DATASPACE. An access mode of R (for read) should be used when your program only reads from variables in the COMMON block. This is particularly important if you wish to share the data file with other users who might access it at the same time.

Note that if more than one COMMON block is mapped onto the same file then all must use the same access mode. This means that if any one of them needs W (write) mode then all must use W mode.

19.5.7 Mapping onto members of partitioned files

It is possible to map a COMMON block onto a member of a partitioned file, using an offset within the member if required. There is a restriction that in this case you are only allowed to use R (read) as an access mode. This is to reduce the risk of a faulty program corrupting other members in the same partitioned file.

There is no need to specify the access parameter in the DATASPACE in this case because READ is the default when a member of a partitioned file is used.

For example:

```
Command: DATASPACE GROUP,ERCC76:FACULTY_GROUP,1024
```

19.6 STORE MAPPING AND PROGRAM PORTABILITY

It is important to appreciate that the facilities described in this chapter are specific to EMAS. When working on programs that are likely to be moved to other systems you should ascertain whether similar facilities exist there. On the other hand, for programs written specifically for use on EMAS the use of direct mapping merits serious consideration.

19.7 CONCLUSION

One of the problems involved in using mapped files lies in overcoming the conceptual block to the idea that a program can access a file without using read and write routines. This difficulty may exist because programmers have grown accustomed to thinking in terms of reading and writing data between the computer system and the 'outside world', even when that outside world is part of the computer system, such as a disc file. Store mapping exploits the fact that, in EMAS, a disc file

can be equated with storage space used by a running program for holding variables.

At first the direct mapping of files seems to be a complicated extension - in fact it is a real simplification. The advantages of accessing data by direct mapping are:

* Simplicity of associated programming: mapping can be regarded as a method of saving data held in arrays from one use of a program to the next.

* Efficiency: mapping avoids the overhead of copying data between a file and a user's data area.

* No limitations imposed by file formats: a mapped file can consist of one byte, or many thousands of bytes structured in a way that is convenient to the programmer.

CHAPTER 20
PARTITIONED FILES

20.1 History of PARTITIONED Files

Partitioned files were introduced on an earlier version of EMAS to meet a number of requirements:

* To allow users to circumvent the problem of being allowed very few files.

* To allow users to group related files together.

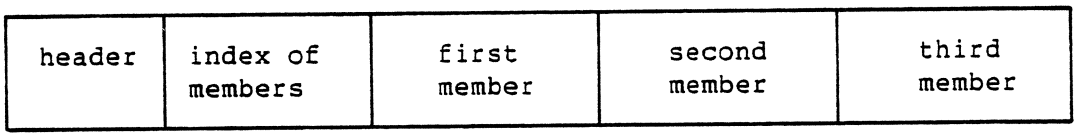
* To provide an efficient method of storing a number of very small files.

The hierarchical filestore provided on EMAS-3 together with the more generous allocations of files and filespace now available means that partitioned files are no longer needed to the same extent to overcome the first two problems. However, in some situations they are still useful. The rest of this chapter describes how to create and use partitioned files, and provides guidance as to the situations in which you should use them.

20.2 The structure of PARTITIONED files

PARTITIONED files share many of the attributes of other EMAS files. The Director (see Appendix 1.3.2) does not distinguish them from other types of file. They are named in exactly the same way as other files. They can be PERMITed, CHERISHed, OFFERed and ACCEPTed exactly as other files. The Subsystem, on the other hand, provides facilities to treat one partitioned file as if it was a large number of other files. This is most readily understood if you consider the structure of a partitioned file. Like other files it has a header, which contains information to say that it is a partitioned file, and the time and date when it was last altered. It also contains a directory, or index, of its members. The rest of the file contains the members themselves. Each of these is a copy of a complete file, itself containing a header and a user data area. These files may be CHARACTER files, OBJECT files, DATA files, even PARTITIONED files.

The structure of a complete partitioned file is:



For certain purposes the Subsystem allows you to access individual members as if they were separate files. In general you can read from individual members, but there are limitations on writing to them.

20.3 Creating a PARTITIONED file

The command NEWPDFILE is used to create a partitioned file. It takes one obligatory parameter, the name of a file. A file of that name must not exist already. If you want to create a file to hold a number of files of data from a series of experiments

you might type:

```
Command: NEWPDFILE EXDATA
```

Having created the partitioned file, you should CHERISH it if you are going to use it to contain valuable information.

20.4 Naming individual members of a PARTITIONED file

An individual member of a partitioned file must have a name consisting of not more than 11 alphanumeric characters, of which the first must be a letter. As with filenames the case of the letters is not significant. An individual member in the partitioned file EXDATA could have a name such as:

```
D230586
```

and to access it you would use the name:

```
EXDATA_D230586
```

20.5 Creating and altering members of PARTITIONED files

The standard method of creating a member of a partitioned file is to use the command COPY. This can take the name of a member of a PARTITIONED file as its second parameter. So to create the member in the example above you could type:

```
Command: COPY DATA2305,EXDATA_D230586
```

As with normal use of COPY, if you copy to a member which already exists then the latter will be overwritten with the new copy.

The two editors described in Chapter 5 both allow direct access to individual members of a partitioned file, both for creating and altering members. They achieve this by editing within a temporary file and then copying the resulting file by using COPY. For example, if a member D230586 did not already exist you could type:

```
Command: EDIT EXDATA_D230586
```

and use the editor to insert the contents.

And, to alter a member D240586 which does exist you could type:

```
Command: ECCE EXDATA_D240586
```

Note that apart from using COPY and the editors you cannot normally create or alter the contents of members of partitioned files. Some utilities and packages allow you to do so by calling the COPY command themselves. Details will be found in the documentation for the utility or package concerned.

In particular you cannot use DEFINE to assign a logical channel to a member of a PARTITIONED file if your program uses that channel for output.

20.6 Reading from members of PARTITIONED files

All Subsystem commands and most of those provided by other users allow you to read from members of partitioned files. Here are some examples:

```
Command: IMP PROGS_VERS1,VERS1Y

Command: LIST EXDATA_D250686,.LP

Command: ANALYSE EXDATA_DA230385
```

You can use DEFINE to assign a logical channel to a member of a PARTITIONED file so long as your program only uses that channel for input. This includes the use of Direct Access and Store Mapped files, which can be read from but not written to.

20.7 Destroying and Renaming members

The DESTROY and RENAME commands can be used in respect of individual members of partitioned files, as well as for the whole partitioned file. Hence the following are valid:

`Command: RENAME EXDATA_D250685,EXDATA_D250686`	Rename a member
`Command: RENAME EXDATA,EXDATA86`	Rename whole PD file
`Command: DESTROY EXDATA_D311285`	Destroy a member
`Command: DESTROY EXDATA`	Destroy whole PD file

20.8 PARTITIONED files within PARTITIONED files

As explained above, each member of a PARTITIONED file has a type, e.g. CHARACTER or OBJECT. It is also possible to have a member of a PARTITIONED file which is, itself, a PARTITIONED file. It is created by using COPY:

```
Command: COPY EXDATA,EXPERIMENT_DATA
```

It is possible to access individual members of the member `DATA` e.g:

```
Command: LOOK EXPERIMENT_DATA_D250586
```

You can repeat this process to many levels if required. The COPY, EDIT and ECCE commands can be used to create lower level files automatically, without you having to make explicit calls on NEWPDFILE, so long as you have created the outermost PARTITIONED file. So you can create a new PARTITIONED file:

```
Command: NEWPDFILE PD
```

and immediately copy a file into an inner PARTITIONED member:

```
Command: COPY TEST,PD_Y1986_MARCH_D03
```

COPY will create the PARTITIONED members `Y1986` and `MARCH` automatically. The same arrangement applies to EDIT and ECCE.

20.9 Relationship between PARTITIONED files and File Groups

The contents of files are not relevant to the hierarchical file system. It follows that members of partitioned files are not recognized as part of the hierarchy. A whole partitioned file can have its place in the hierarchy, of course, just like any other file, but its subdivision into members is not 'visible' in the hierarchy. Partitioned files are available in EMAS-3 as a convenient facility for the internal organization and subdivision of files. They are not intended to provide facilities which emulate, or are compatible with, the organization of whole files into groups.

Partitioning should be seen as a way of subdividing or organizing the data within a file. The list below indicates the criteria that should be considered when choosing between partitioned files and groups of files.

- File groups are more suitable when individual members are subjected to frequent changes.
- File groups must be used if different CHERISH status or access permissions are needed for individual members.
- File groups must be used if the files are to hold direct output from programs.
- Partitioned files are suitable when there are large numbers of members (more than 20) whose contents generally remain static. They are also more suitable if you have many very small members or when you need to access many members at the same time.

The two commands GROUPTOPD and PDTOGROUP simplify the task of transferring information between PARTITIONED files and File Groups.

20.9.1 The command GROUPTOPD

GROUPTOPD copies the contents of a group into a partitioned file:

```
GROUPTOPD group, pdfile, compress
```

The first parameter is the name of a group, which must exist and be accessible for reading.

The second parameter is the name to be given to the partitioned file produced from the specified group. A file of this name must not exist already.

The third parameter is optional. It specifies what you want to happen to groups which are members of the group you are copying. If there are none then this parameter is ignored. This parameter should be set as follows:

n if you do not want sub-groups to be copied into the partitioned file.

y (the default) if you want each sub-group to be copied into a separate partitioned file and each of these partitioned files copied into the main partitioned file. This process will continue with sub-sub-groups etc. thus producing a partitioned file with a structure which is equivalent to that of the original group.

20.9.2 The command PDTOGROUP

PDTOGROUP copies the contents of a partitioned file into an equivalent group.

```
PDTOGROUP pdfile, group, expand
```

The first parameter is the name of a partitioned file, which must exist and be accessible for reading.

The second parameter is the name to be given to the group produced from the specified partitioned file. A group (or file) of this name must not exist already.

The third parameter is optional. It specifies what to do with any members which are, themselves, partitioned files. If none of the members are partitioned files then this parameter is ignored. This parameter should be set as follows:

n if you want to treat members which are partitioned files in the same way as all other members - that is, left as partitioned files in the group.

y (the default) if you want each of these members copied into a sub-group containing separate files for each of their members. (And so on as far as the nesting continues.) If this option is selected the resulting file group will have a structure which is logically the same as the original partitioned file.

CHAPTER 21
MAGNETIC TAPE FILE HANDLING

21.1 INTRODUCTION

Magnetic tapes on EMAS fall into two categories. First there are the tapes that the System uses on your behalf to maintain the backup and archive file stores (whereby the integrity of your CHERISHed and ARCHIVEd files is maintained). While you indirectly invoke the use of these tapes, you are relieved of all need to know anything about them because such tapes are an integral part of the System. Secondly there are what can be called private user tapes - tapes which the user owns, accesses directly and of which he has (and is obliged to have) a detailed knowledge.

There are three groups of users who are likely to need to use private tapes:

* Those who import data from other installations. If you are in this category you may find it easier to use the File Transfer mechanism which is introduced in Section 15.10. This provides access to a large number of computing services in universities and Research Institutions.

* Those who export data to other installations. Again you may find that the File Transfer mechanism is more convenient.

* Those who handle files which are larger than normally allowed in the EMAS filestore. If you are in this category you should discuss your requirements with the System Manager in order to determine whether special arrangements can be made.

In order to enable you to deal with all the unpredictable situations that such tapes can bring, the access to these tapes must remain as flexible as possible. This means that you must be reasonably acquainted with the technicalities of handling tapes and that the bridge between the tapes and EMAS has to be maintained by you, usually in the form of parameters to the commands involved with tape handling. Every effort is made to ease this inconvenience but it cannot be completely avoided.

Unless there is a necessity for using tapes, you should use the EMAS file system (and associated archive facilities) for the storage of data. The use of tapes has several disadvantages:

* Integrity. CHERISHed files on the EMAS file system are regularly backed up and ARCHIVEd files are recorded twice; hence the overall integrity of user data is well protected. Your user tapes are under your control and unless you take exceptional care the integrity of data on such tapes cannot be so well protected as that in the file system.

* Access. Even if an installation allows interactive access to tapes, there is generally a delay while a tape is made available.

* Inconvenience. As has been mentioned above, the user of tapes has to be aware in a detailed sense of what using magnetic tapes involves, remembering the contents of such tapes, the details of file structures, etc.

* Cost. The cost of processing data from tapes is greater than from the file system because of the extra software, buffers, paging etc. that handling tapes involves. Because of the nature of file handling in EMAS where data in the file system is concerned such overheads are not incurred. Some installations may impose a surcharge each time a tape is used.

21.2 Obtaining and Registering User Tapes

If you wish to obtain a tape for private use (i.e. for access as described in this chapter) then you should obtain an application form from the Advisory Service or from Job Reception and submit it to ERCC with the appropriate order form. If you already have a tape which you wish to use on EMAS you will need to register it with the operations staff. Again, you should obtain a "User Magnetic Tape Form" from the Advisory service.

When a new tape is first used it has to be labelled. This involves the computer operators running a special program to write a label at the beginning of the tape. It is important that this label is written at the correct density, since this will determine the density at which information will be stored on the tape for the rest of its life (or until it is labelled again). If you propose to use the tape only on EMAS-3 then you should specify the highest density available - 6250 bits per inch. If you propose to use the tape elsewhere then you should determine what density of tape can be used at the other installation. You cannot mix files recorded at different densities on the same tape.

21.3 STANDARDS

21.3.1 Hardware Standards

Installations running the EMAS-3 System will normally be able to process tapes having the following characteristics:

No. of tracks	9
Width	0.5 inch
Density	1600 or 6250 bits per inch (bpi)
Parity	Odd

Some installations may also be able to process 800 bpi NRZI tapes.

21.3.2 Identification

Before data can be retrieved from a tape the tape has to be found; this generally involves an adhesive label on the tape reel. Such labels should not be put on a case enclosing the tape reel; indeed, you should not attach adhesive labels yourself but leave this to operators who have labels with suitably strong adhesive: adhesive labels that come off tape reels (especially when a tape is being rewound at high speed) can cause considerable damage to tape decks and invariably ruin the tape. The identification for the tape reel as written on the adhesive label should match that in the volume label recorded at the start of the tape (if there is one).

Once a tape has been found, it is mounted on a tape deck and then the System tries to identify it by attempting to read a magnetic label recorded at its start. This label, known as the volume label, is checked against the volume identification supplied by the user accessing the tape and the access request is rejected if the check fails. This affords a degree of security against the possibility of a tape being overwritten by accident. Thus you should try to ensure that tapes that you import into any installation have volume labels; some installations will not allow direct user access to tapes that do not.

It is important to understand the distinction between the two aspects of labelling magnetic tapes.

* First, there is a paper label which is stuck on the tape reel. This may identify the owner and origin of the tape and should show the volume label of the tape (also known as its serial number).

* Secondly the tape normally has, as its first recorded block, a VOL label. This is written on the tape, by the operator, when it is new, and usually stays there for its life. This VOL label is read by EMAS automatically as soon as the tape is mounted on a tape deck. It is this information which is used by EMAS to identify your tape. To avoid confusion the VOL label on the tape should be identical to that on the paper label. If you are importing tapes from other installation you should check that this is the case.

21.3.3 Software Standards

Data on magnetic tape is usually recorded in a format that is defined by one of several standards.

The standard adopted for use on EMAS is that defined by IBM, because this is well documented and the most widely understood standard available. The definition of the labels recorded on IBM standard labelled tapes is given in the IBM publication

OS/VS Tape Labels (ref GC26-3795)

and the definition of the various record and block structures that can be used within tape files is given in another IBM publication

OS/VS2 MVS Data Management Services (ref GC26-3875)

21.4 HIGH LEVEL LANGUAGE ACCESS

Files on magnetic tapes can be processed directly by user programs. The most secure and generally most convenient tape structure to use is that conforming to the IBM labelled tape standard (the default EMAS standard). However, tapes not conforming to this standard can be processed in unlabelled mode, i.e. you can request access to physical files.

To access tape files a program normally uses the input/output facilities relevant to the language in which the program is written; these in turn use a logical channel number to refer to the required file. To establish the correspondence between a particular file on tape and such a logical channel, you must call the Subsystem command DEFINEMT.

```
DEFINEMT channel, file, vol, label, record format, blocksize
```

`channel` An integer normally in the range 1-1000. This specifies the channel number used by the program to gain access to the file. Note that only one file per tape can be selected by your program for reading or writing at one time, although more than one can be defined. This parameter is mandatory.

`file` The name of the file on the tape - obligatory when writing new files and strongly recommended when appending to existing files. The name of a file as recorded in the label on tape can be up to 17 characters long. If a longer name is specified then it is truncated to 17 from the *left*. For IBM compatibility the required form is:

```
name1.name2.name3 ...
```

where `name1` must be given and each of `name1`, `name2`, etc. consists of not more than 8 upper case letter or digits, starting with a letter. As few or as many names as required, separated by dots, may be specified, subject to the overall maximum of 17 characters. In particular, the name does not have to be in the standard EMAS-3 format. This feature may be needed when reading tapes on EMAS which were written at other installations.

`vol` — The tape volume (i.e. serial) number. If it is followed immediately by an asterisk the tape will be loaded with a write permit ring fitted. This is essential if it is intended to write to the tape. Otherwise no write ring will be fitted and the tape will thus be protected from inadvertent overwriting.

`label` — A positive integer which specifies the position of the file on the tape (default=1 when reading; no default when writing).
Note that writing to a tape file with label `n`, say, automatically destroys all files on the tape with labels greater than `n`. Thus, if one writes a file with label 2 on a tape which previously contained 5 files, at labels 1 to 5, the effect is that the label 1 file is unchanged, the label 2 file is overwritten, and the files at labels 3, 4 and 5 are all lost.
If it is required to modify an existing file by appending data to it, i.e. adding data to the end of the file, then this parameter should be suffixed by the characters "`-MOD`". See also "Unlabelled Access" below.

`record format` — Of the form "`rn`", where

- `r` is the format: one of F, FB, V, VB, VBS or U.
- `n` when the format is "variable", the *maximum* number of bytes per record available for user data; when the record format is "fixed", the *actual* number of bytes available

`blocksize` — A positive integer in the range 18-32760. It is used when writing files with fixed length records to specify the length of each block, and when writing files with variable length records to specify the maximum length of each block.
Note that if a fixed record length is used, the blocksize must be an exact multiple of the record size; if a variable record length is used, the blocksize must be at least 8 bytes greater than the record size.
(Default when writing = 4096, or a value close to 4096 compatible with the `record format` parameter.)

Note: The `format` and `blocksize` parameters can be used when reading tapes to override the values found in the file labels - useful when processing tapes with incorrect labels.

21.4.1 Unlabelled Access - Processing Physical Files

A tape that does not conform to the accepted system standard can be processed in unlabelled mode. In this mode no assumptions are made about the layout of the tape other than that access is to physical files, i.e. collections of data blocks delimited by tape marks. This mode is invoked by prefixing the label parameter with the letter "`P`". The file name parameter is ignored and the format and blocksize parameters are obligatory.

Implications:

* The label parameter identifies a physical file.

* Writing to the first physical file on a tape is not permitted - the first physical file normally contains a volume label.

Examples of valid calls on DEFINEMT

```
Command: DEFINEMT 7,GRADE.TESTXY,AS1357
```
or
```
Command: DEFINEMT 7,,AS1357
```
Reading labelled file 1

```
Command: DEFINEMT 18,SAMPLE,EA3265*,12,FB100,2000
```
Writing labelled file 12

```
Command: DEFINEMT 5,,XF4567,P20,VB220,4000
```
Reading physical file 20

21.4.2 Successive Processing of Tape Files

By default, each time a tape file is closed the tape is rewound to BT and unloaded to release the tape drive for further use. If, however, you wish to process several files on the same tape, this results in the tape having to be reloaded for each file, causing undesirable delays. To avoid this, a command HOLDMT is provided to hold the tape at its last position. This command takes one parameter, the name of a tape. E.g.

```
Command: HOLDMT AB1234
```

HOLDMT should be called before the first file in a processing sequence is accessed.

When a held tape is no longer required, you should call the converse command RELEASEMT which unloads held tapes to free the tape drives for further use. If this command is not used, all held tapes remain on the tape drives until your job or session ends, thus unnecessarily denying use of the drives to other users. This command takes the name of a tape or ".ALL" as its only parameter. E.g.

```
Command: RELEASEMT AB1234
Command: RELEASEMT .ALL
```

21.4.3 Character Codes

Character information on tape (e.g. written using formatted I/O in Fortran) is generally held in EBCDIC representation. Since EMAS normally uses ISO for its internal representation, character I/O to or from tape involves an ISO to EBCDIC translation when writing to tape and vice versa when reading from tape. Note that this translation is only carried out in the case of Fortran formatted I/O. Fortran unformatted and IMP sequential access do not involve translation.

21.4.4 Error messages

These are detailed below, after the section "Utility access".

21.5 UTILITY ACCESS

The utilities described in this Section are for processing users' tapes written to recognizable standards, *not* tapes used by the System for its own private purposes (such as archiving). You should bear in mind that although user tapes can be processed on EMAS, the standard EMAS archive system provides comprehensive facilities for long term storage of data.

The utilities described below have been designed to be complementary in respect of the facilities they provide, although their modes of operation are different. In particular, the utilities involved in copying files from tape to the EMAS filestore and vice versa are *not* symmetrical, and their parameter lists differ.

The utilities can in principle be used from foreground (interactive) or background (batch). However, access to user magnetic tapes from a foreground process is not normally permitted; consult your local Advisory service for details.

21.5.1 The command TAPEANAL

This utility is used to scan a tape and print out a summary of its contents, the level of detail in the summary being controlled by the LEVEL parameter. It should be noted that reading every block on a tape can be an expensive process. Using the LEVEL parameter set to 1 or 4 avoids this problem. The values 2, 3 and 4 of LEVEL are normally used for diagnostic purposes.

```
Command: TAPEANAL tape, output, level, dump
```

Parameter	Default	Description
`tape`	none	The name of the tape to be analysed. This parameter must be specified (all the others are optional).
`output`	.LP	Destination for the summary output from the analysis. This can be an output device or a file.
`level`	1	The type of analysis obtained. Values 1 and 2 assume that the tape conforms to the System standard. 1 A scan and extraction of information from the labels on the tape. This gives a summary of the characteristics of all the files on the tape. 2 As 1 but including a scan of the files as well as the labels, thus checking for I/O errors and reporting the longest block in each file. This is less efficient than 1. 3 This option assumes nothing about the structure of the tape but simply gives a block by block summary of each physical file on the tape. 4 Scans and formats all blocks that resemble labels, either by their positions on the tape or by their structure. This is useful when dealing with incorrectly written labels.
`dump`	null	If a dump of the first 96 bytes of the first three blocks of each file is required then this parameter must specify the file for the dump output.

Examples

1) To obtain a simple summary of the contents of tape AB1234:

```
Command: TAPEANAL AB1234
```

2) To obtain a complete summary of the same tape, including a dump of the first 96 bytes of the first three blocks of each (labelled) file on the tape, and then to print the summary and dump together:

```
Command: TAPEANAL AB1234, SUMRYFILE, 2, DUMPFILE
Command: LIST SUMRYFILE+DUMPFILE, .LP15
```

Notes

1) Although the headings on the output for a level 1 analysis are reasonably self explanatory, the following notes may help:

   Block Length: the maximum permitted size for blocks in the file.

   Label No.: the file sequence number that appears in the file label (this need not necessarily start at 1 for the first file on the tape).

2) For a level 2 analysis, three extra fields are printed:

   Max Block: the length of the largest block actually found in the file.

   Byte Count: the total count of all bytes in the blocks of the file, excluding the file labels but including all 'red tape' for variable format files.

   Length: an estimate of the actual length of tape occupied by the file (including its labels) rounded up to the next foot. This assumes an inter-block gap of 0.6 inch.

3) The output from a level 3 analysis relates to the physical files on the tape. The output for each file consists of the physical file number followed by a number of fields, in the form:

```
BLOCKNO( COUNT * SIZE)
```

   where SIZE is the length (in bytes) of COUNT blocks starting at block BLOCKNO, and taking the first block on the tape as block 1. For example:

```
13( 9 * 1600)
```

   would mean that there are 9 blocks of 1600 bytes starting from and including block 13 on the tape.

4) The output from a level 4 analysis consists of a printout of all blocks on the tape that could be labels - even if incorrectly written. The labels are split up into their various fields (as specified in the IBM publication "OS/VS Tape Labels"), on a single line, and the fields are identified on the following line.

5) For levels 1 to 3, you may restrict the output to 72 characters per line by prefixing the value for the parameter by the letter 'S'. For example:

```
Command: TAPEANAL AB1234,ABANAL,S1
```

21.5.2 The command FROMTAPE

This utility is used for copying files into a user process from IBM standard labelled tapes, or from other tapes by processing them in unlabelled mode. Files can be processed singly but facilities are provided for more complex tasks.

```
Command: FROMTAPE tape, input, output, translate
```

`tape` The name of the tape from which the files are to be copied. This parameter must be given; the others are optional.

`input` The label position of (and format information about) the input file. The simplest form of this parameter is

```
n
```

The general form is

```
Pn(record format, blocksize)
```

This parameter is optional. Its components, which are each optional, are as follows:

`n` The label position of the input file, by default 1. If prefixed by `'P'`, this indicates that "physical" (unlabelled) processing is required; i.e. the nth *physical* file on the tape is specified. See also "Multiple processing" and the examples, below.

The section in brackets is optional. If given, it specifies values for one or two subparameters:

`record format` The record format of the tape file. Default: for labelled processing the information is taken from the input file label; for physical (unlabelled) processing the default is FB80.

`length` The maximum or fixed block length. Default: for labelled processing the information is taken from the input file label; for physical (unlabelled) processing the default is 4800.

`output` Specifies details about the EMAS disc file(s) to be written. The format of this parameter is

```
name(type, size)
```

This parameter is optional. Its components, which are each optional, are as follows:

`name` The name of the output file, by default derived as explained in "Derivation of output file name(s)", below.

The section in brackets is optional. If given, it specifies the values of one or two subparameters:

`type` This can take the following values:

`CHAR` Output will be to a character file with EBCDIC to ISO translation, all trailing spaces removed and a newline character added to the end of each record. This is the default value.

`CHARX` As `CHAR` but with no newlines added.

CHARA — As CHAR but with ANS control character interpretation if appropriate.

DATA — Output will be to a variable format data file, except in the case of fixed format input records where it will be to a fixed format data file.

size — This field has the form

```
Kbyte*count
```

where Kbyte is the size of the output file in units of 1024 bytes, and count is the number of component output files of that size into which the input file is to be split. The utility will stop if the total output allocation specified by the two parameters is exceeded. The default for Kbyte is 255 and for count is 1. For example:

```
50*3
100*
```

translate — An optional fourth parameter is the name of a file to be used as a translate table to be applied when the output type is one of the character variants; by default a system supplied table is used which translates from IBM EBCDIC to EMAS ISO; if specified as ".NULL", translation is suppressed, otherwise the first 256 bytes of the nominated file are used as the translate table; the translation is assumed to be to ISO in the sense that trailing space deletion removes ISO spaces after translation; the null option will be convenient for tapes loosely described as 'in ASCII format' which often contain ISO card images.

Prospective users of the translate table facility should read the following description of the mechanics and hazards involved.

A translate table can be described in IMP terms as a byteintegerarray(0:255). The character to be translated has a binary value also 0-255; it is replaced by the character in the array element indexed by that binary value.

When translating from ISO, as in TOTAPE, use of the translate-table-file facility implies that the user is translating to some non-standard character code, so no sensible checking can be done on the result of the translation. When translating (nominally) to ISO, as in FROMTAPE, it could be argued that the result of the translation should be within the ISO subset, but this would constrain the sophisticated user who may wish to translate to anything he chooses, perhaps with a view to further processing. Allowing this freedom, there is no reason even to suppose that the translation should not be many-to-one. In practice, no checking whatsoever is performed on the result of the translation.

The file which contains the table is accessed in a very basic way. The type of the file is irrelevant, and so long as it has an even partially valid subsystem header, the translation will proceed without comment. In short, any file in the entire EMAS system, which exists and is accessible in read mode can be successfully(?) used, its first 256 bytes effecting the translation.

Clearly there is great scope here for disaster! A wrongly nominated file will cause a translation using an arbitrary 256 bytes (of object code?, source code?, text?); it is very likely to be many-to-one, so will not be reversible when(if) the error is discovered; if the original file has been

destroyed in the interim, all is lost. In the light of experience, some constraints may have to be applied, but in the meantime, this facility must be used with great caution.

It will be relatively trivial to add often-used tables to the system supplied set, to be referenced by keyword.

Summary of defaults

```
Command: FROMTAPE AB1234
```

In this case the file at label number 1 on magnetic tape AB1234 is copied to produce an EMAS disc character file, with a name derived from that of the input tape file. Its maximum permissible size is 255 Kbyte.

```
Command: FROMTAPE AB5678,P4
```

In this case the 4th physical file on tape AB5678 is copied to produce an EMAS disc character file, with name T5678P04F (see "Derivation of output file name(s)", below). Its maximum permissible size is 255 Kbyte. The file on tape is assumed to have a record format of FB80 and a blocksize of no more than 4800.

Simple examples

1) To copy the 4th file from standard labelled tape AB1234 to a character file with a name derived by the utility:

```
Command: FROMTAPE AB1234,4
```

2) To copy the 11th physical file (variable length records up to 193 bytes long in blocks up to 4000 bytes long) from tape XY6789 to a data file called SAMPLE1:

```
Command: FROMTAPE XY6789,P11(VB193,4000),SAMPLE1(DATA)
```

Derivation of output file name(s)

If you do not specify a name for the output file the utility will derive a name. For labelled files this is the rightmost (perhaps only) field of the input file name.

For physical files a name is constructed of the format

```
TvvvvPnnF
```

where 'vvvv' are the last four characters of the name of the tape and 'nn' is the label number of the file.

If several component output files are produced from a single input file, the output files are named in sequence. For example:

```
OUTPUT=FRED(DATA,74*3)
```

would produce files FRED, FRED2, FRED3. This scheme of generating file names also applies when a list of files is being processed (as explained below in "Multiple processing") and the name field of the output parameter has been specified. Example (4) below should clarify this.

Multiple processing

There are two ways in which the utility can be made to process several files at a time. First the label field of the input parameter can specify a list of individual labels or ranges of labels, e.g.

```
INPUT=(2,5-11,23)
INPUT=P(31-45,62)(FB80,4000)
```

Secondly you can instead specify as input a file, say CFTINST, that contains a list of further instructions to the utility, e.g.

```
INPUT=@CFTINST
```

The instructions must have the same format as the parameter list to the utility itself but excluding the TAPE parameter. For example, using positional parameters:

```
3,OUTFILE
(5-13,17,23-30),(DATA,100*)
(41,47,49-52),(CHAR)
```

or

```
P(11-18,23-27)(FB80,3200),(CHAR)
P(35-38)(VBS20008,4000),(DATA)
```

The effect of mixing labelled and unlabelled processing in any one call on this utility is undefined.

As a result of the label list facility, the full format of FROMTAPE's parameter list becomes:

(tape,null or P(label list)(format,blocksize),filename(type,size*count))

More complex examples

3) To copy files 2, 5 to 11 inclusive and 23 from standard labelled tape AB1234 to data files in your process:

```
Command: FROMTAPE AB1234,(2,5-11,23),(DATA)
```

4) To copy physical files 7, 23 and 42 to 59 inclusive, each having record format FB80 and blocksizes less than 4800, from tape XY6789 to character files named XYFILE, XYFILE2, ..., XYFILE20:

```
Command: FROMTAPE XY6789,P(7,23,42-59),XYFILE
```

21.5.3 The command TOTAPE

This utility is used to copy character or data files from a user process to a private magnetic tape. The tape can be written in standard (IBM) labelled format or in unlabelled mode. Files can be processed singly but facilities are provided for more complex applications.

```
Command: TOTAPE tape, label, input, output, translate
```

`tape` The name of the tape to which the files are to be written.

`label` The label position at which copying is to start. If unlabelled (i.e. physical) files are required, then this should be prefixed by the letter 'P'.

input The name of the file to be copied to the tape. If several files are to be processed then this parameter should specify the name (prefixed with '@') of a file in the user process containing instructions to the utility. See "Multiple Processing", below.

output The format of this parameter is

```
name(format,blocksize,option)
```

All the fields of this parameter, and hence the parameter itself, are optional.

name The name to be given to the file on the tape. If this is omitted then the name will be that of the input file.

format The format in which the records will be written to the tape. This is defaulted to FB80 for character files and to the format of the input file for data files.

blocksize The size of the blocks in which the data will be written to the tape. See DEFINEMT (Section 21.4).

option For character files this specifies what action should be taken if an input 'line' is found that is longer than the output record length. It can take one of three values:

STOP Processing is abandoned.

SPLIT The 'line' is split across two or more output records.

TRUNC The 'line' is truncated.

translate An optional fifth parameter is the name of a file to be used as a translate table (relevant only for character files); by default, a system supplied table is used which translates from EMAS ISO to IBM EBCDIC; if this parameter is specified as ".NULL", translation is suppressed, otherwise the first 256 bytes of the nominated file are used as the table; the translation is assumed in all cases to be from ISO, only to the extent that if fixed length output records are required, trailing ISO spaces are inserted prior to translation.

Prospective users of the translate table facility should read the warning in the description of the parallel parameter in FROMTAPE.

Examples

1) To copy character file TEXTONE to labelled file 14 on tape AB1234 using default record format and blocksize:

```
Command: TOTAPE AB1234,14,TEXTONE
```

2) To copy a data file CENX with variable length records up to 25000 bytes long to the same tape at the 15th file position and giving the output file the name CENSUS81.SUMMARY:

```
Command: TOTAPE AB1234,15,CENX,CENSUS81.SUMMARY(VBS25000)
```

In this example it is necessary to specify the output format because the format of the input file is not practicable for tape files.

3) To copy a character file SOURCEX to the 12th physical file on tape LL5678 with format FB80, blocksize 2000 bytes and truncating any 'lines' longer than the output record length of 80:

```
Command: TOTAPE LL5678,P12,SOURCEX,(,2000,TRUNC)
```

Note that the default output record format of FB80 corresponding to a character input file has been taken.

Multiple Processing

Copying several files at a time is a frequent requirement and for several reasons it is better if this is done by using only one call on the utility. For this purpose you can specify a list of instructions to be processed by editing such a list into a file and specifying the file to the utility via the "input" parameter but prefixing the filename by the '@' character (as explained above). The instructions, one per line, should take the same format as the parameter list to the utility but omitting the "tape" and "label" parameters. The tape label number is incremented for each successive output file. For example, to combine the first two examples given above, edit a file, CTTLIST say, to contain the text

```
TEXTONE
CENX,CENSUS81.SUMMARY(VBS25000)
```

Then call the utility as follows:

```
Command: TOTAPE AB1234,14,@CTTLIST
```

A partitioned file of character or data file members can be copied in its entirety, the names of the files written to tape being those of the members. If a partitioned file is given as input then all the suitable members (e.g. *not* object or directory files) will be individually copied, the names given to the files on tape being the names of the members. If the name parameter is used in these circumstances, the member name is prefixed by the value specified for name (e.g. name.member). This is particularly useful when writing out several partitioned files containing members of the same name. If a single member is specified as input and the name parameter used, then the name used on the tape will be just that specified by the name parameter.

Use of multiple processing has the advantages of only making one call on the utility, only one request for the tape, and the output is tabulated for all the files copied in a form that is more convenient for the user than outputs from separate runs of the utility.

Note

Combining labelled and unlabelled processing with this utility is not recommended. In any event writing in an unlabelled mode to a labelled tape can lead to that tape being unfit for subsequent labelled access. You are not allowed to write to physical file 1 by using this utility, since this is the location of the tape label. This can only be changed by a privileged program which has to be run by the operations staff.

21.5.4 The command COPYTAPE

This utility can be used to copy complete tapes or a selection of files from input tapes to a single output tape. The tapes can be copied as tapes conforming to a recognizable standard or as unlabelled tapes.

```
Command: COPYTAPE input, output, standard, erropt, blocksize
```

Parameter	Default	Description
input	none	This specifies the input to the utility, which can take one of three forms, as follows: `tape name` All the files on the specified tape are copied to the output tape. `tape name(label list)` The files specified are copied to the output tape. The label list must consist of label numbers or ranges of label numbers, separated by commas. For example: `AB1243(3,6,11-23,45)` `@filename` A file name (prefixed by '@') which contains a list of instructions, one per line, of the same format as above.
output	none	The name of the output tape and, in brackets, the label at which the output is to start. Note that there is no default label number.
standard	IBM	The standard to which the tapes are to be processed. At the time of writing this parameter can take one of three values: `IBM` Tapes written to the accepted System (i.e. IBM) standard. `UL` Unlabelled. Each physical file is copied and no attempt is made to interpret any of the blocks as labels, with the exception of the first file (which conventionally holds a Volume Label). You are not allowed to write to the first file of an unlabelled tape. A double tape mark is deemed to mark the end of recoverable data on a tape. `SDTMn` where 'n' is an integer; this is the same as UL above, except that a double tape mark does not mark the end of the data; instead the double tape mark limit is set to 'n'; lesser double tape marks are treated as single tape marks.
erropt	STOP	Error option. This specifies the action to be taken in the event of an I/O error being encountered while reading an input tape. `STOP` Processing stops. The utility will attempt to terminate the output tape correctly. `SKIP` The block in error will be skipped. Note: this option is not recommended for use with files containing spanned records.
blocksize	10000	The size of the largest block to be processed.

Examples

1) To copy the contents of standard labelled tape AB1234 to tape XY5678:

```
Command: COPYTAPE AB1234,XY5678(1)
```

2) Using the same tapes as in (1) but selecting the files from the input tape in the order 15, 10, 11, 12, 13, 5 and starting the copying at label 4 of the output tape:

```
Command: COPYTAPE AB1234(15,10-13,5),XY5678(4)
```

3) To copy a selection of files from several tapes, say AB1234, LM5678, XY4444, to tape PQ3333. First edit a file, say MULTSEL, to contain the text:

```
XY4444(5-8,3,9,2)
AB1234(3,1,5,19-34)
XY4444(23-30)
LM5678(5)
```

Then call the utility:

```
Command: COPYTAPE @MULTSEL,PQ3333(1)
```

21.6 ERROR MESSAGES FROM RECORD LEVEL TAPE HANDLING

The following messages may appear when processing files on magnetic tape, either by means of a user program (and DEFINEMT) or by use of FROMTAPE or TOTAPE. The explanations given are the most likely but not necessarily the only ones. The calling procedure will stop with diagnostics if any of these error conditions arises.

Invalid tape format

The tape does not conform to the accepted System standard and cannot be processed except in unlabelled mode (or perhaps by specialized utilities).

File header error

One of the file header labels is not written to the appropriate format or is missing. Try a level 4 tape analysis to diagnose the problem. Unlabelled processing will probably avoid the difficulty.

Label number error

The file sequence number recorded in the HDR1 label of the file is not consistent with the position of the file on the tape. This can arise if a file has been incorrectly copied from one tape to another. Use unlabelled processing.

File name error

The file name provided by the user does not match that recorded in the HDR1 label for the file at the requested position. Resubmit work with file name correctly specified.

Failed to create buffer

The tape handling routines have failed to create a temporary file to use as a buffer to process the tape file. This usually occurs because your file index is full. Freeing about 100 Kbyte of file space normally cures the problem.

Record format incorrect

When processing a spanned record file, an inconsistency has been found in the structure of a record. This usually happens at the end of a file, when the last block does not finish with a completed record (perhaps because the process writing the file terminated unexpectedly, e.g. running out of time). It can also arise because of an inconsistency between the information in the file label and the actual structure of the records in the file; if this is the case provide overriding format information.

Tape request refused by Oper

A request to have a tape mounted has been refused, e.g. because the tape was not available or access was not permitted.

Failed to write label

The System has failed to write a file label, a most unlikely occurrence. Contact your local Advisory service.

Failed to position tape

The System has failed to position the tape at the specified label, because it has lost track of where the tape has been positioned, or failed to rewind the tape, or some other unusual circumstance. Resubmit work and if the problem persists contact your local Advisory service.

I/O error

An I/O error has been encountered. The diagnostics will indicate which block in the file is in error. This is not likely to occur when writing. The file in question normally has to be recreated.

Logical end of tape

This message can occur when reading, or positioning a tape prior to writing. It implies that the logical end of tape or end of recorded data has been reached, normally when attempting to find a file at a label position that does not exist, i.e. with a label number greater than that of the last file on the tape. A level 1 tape analysis will indicate exactly what files are on a labelled tape.

Physical end of tape

This message can occur when writing a file. It means that the physical end of tape has been reached, i.e. the tape cannot accommodate any more data. The file being written at the time of this failure is closed with "End of Volume" labels.

Irrecoverable failure

For some unusual reason, such as mechanical failure of a tape deck, the tape is no longer accessible. Resubmit work and if the problem persists contact your local Advisory service.

Max tape channels exceeded

You have requested a greater number of tapes to be simultaneously accessible than is possible with the number of channels permitted.

Tape already in use

An attempt has been made to access a tape which is already in use, i.e. on which there is a file which is being processed. It is impossible to have concurrent access to more than one file on one tape.

Invalid format or blocking

The format or blocking information in the file label does not comply with the standard. Provide overriding information and resubmit.

Record too long

A record has been found that is longer than specified (by the user or in the file label). The user can circumvent the problem by supplying overriding format information.

Invalid unlabelled access

An attempt has been made to write to the first physical file on a tape. As this file is potentially the volume label, such access is not permitted. If you have a genuine requirement for a completely unlabelled tape you should contact your Advisory service in the first instance.

Too many tapes requested

More tapes were requested than specified in the DETACH parameter (see Section 16.3).

Tape already claimed

A tape has been claimed when already allocated to the requesting user.

21.7 IMPORT/EXPORT OF DATA ON MAGNETIC TAPE

When writing data to magnetic tape with the intention of transporting it to another installation, you should bear in mind the following points:

1) It is normally quicker, simpler and cheaper to take advice before embarking on

the use of tapes rather than face the trauma of sorting out problems afterwards.

2) Even if you do not understand all of these notes, the staff at the sending and/or receiving installation will and they should be able to give you suitable guidance. If you are in any doubt as to how to use tapes, contact your Advisory service in the first instance.

3) Use new tapes. Old "soft" oxide tapes or those that have lain around in dusty offices can cause considerable expense and delays, especially when one considers how cheap new tapes are (less than £10) compared with processing costs or even postage!

4) The labels fixed to tape reels use a special quality of adhesive, so *do not* apply your own. If you write on these labels use a felt or fibre tip pen - not a pencil.

5) Always transport tapes in one of the special enclosures in which they are normally stored, using cases in preference to self-loading clips. Further, put them in some sort of insulated packing, e.g. a padded mailing envelope.

6) Ensure that the receiving installation will be *physically* able to read the tape that you propose to write, i.e. that it will be recorded at a suitable density and in a suitable mode.

7) Even if the tape is physically readable, will the recipient be able to interpret it *logically,* i.e. what tape, block and record structures will be acceptable? The most widely understood structures are those defined by IBM.

8) Do not write tapes with blocks longer than about 4000 bytes. This length is considered to be a reasonable compromise between efficiency and readability.

9) If at all possible, record all files twice as insurance against I/O errors. For example, if six files are required write them at labels 1 to 6, say, and then again at labels 7 to 12.

10) A tape from another installation should always be accompanied by at least a summary of the contents of the tape plus a *clear* statement of the standard to which the tape was written, e.g. IBM standard labelled. Note that even if a tape is unlabelled, you still need to know the structure of the records within each block.

11) Many installations have a number of tape utilities that are intended for internal use, generally producing tapes of non-standard formats. While such utilities and tapes may be convenient within the context of a particular operating system, they are generally inappropriate for transferring data between different installations, and they should thus be avoided for this purpose.

12) Beware of transferring data in binary form. Different machines can use different word lengths, arithmetic real quantities can vary in format, and bit and byte inversions are not unknown. Unless you are sure that the machines concerned are compatible in their binary representation of arithmetic quantities, it is safer to transfer numeric data in character form.
N.B. Files containing a mixture of character and binary data can give rise to problems out of all proportion to any imagined advantages.

13) When moving from one range of computer to another, internal representations of characters can cause problems.

Characters to avoid

Newline, Carriage return, Linefeed, Formfeed
Currency symbols
Hash (#), vertical bar, split vertical bar, sub
Any non-visible characters in general
Control characters in general, for example vertical and horizontal tabs, backspace, escape etc.

Characters which are generally safe to use

Upper and lower case letters, numerals, space
+ - / * = & ? @ % () {} < > : ; , . " '

Characters not mentioned in the two groups above may cause problems or may not depending on the compatibility of the character set representations in use at the originating and receiving installations. This area has a long history of difficulties and use of incompatible character sets can involve lengthy remedial work on transported files.

14) Additional documentation such as a character set definition and a specification of any tape labels used (format *and* interpretation) can be of considerable assistance to the receiving installation.

15) The format in which you write tapes will be one which the originating installation can generate and which the receiving installation can accept. However, it is worth remembering that, for program source text and some forms of data, "card image" format (fixed length 80 byte records, blocked to 2000 bytes say), although somewhat antediluvian, is simple and is widely understood. If you have the choice, this is probably the safest format to use.

16) Technical details specific to EMAS services at ERCC:

* Hardware Standard: 9 track tape, 1600 bpi Phase Encoded.

* Tape Labelling Standard. As defined by IBM in their document "OS/VS Tape Labels", ref. GC26-3795.

* Block and Record Structures. As defined by IBM in their document "OS/VS2 Data Management Services", ref. GC26-3875. Although tapes of other formats can be physically read, the interpretation of the data may involve considerable effort and specially written utilities.

* Character data on tape should use the IBM EBCDIC representation.

APPENDIX 1
THE STRUCTURE OF EMAS-3

A1.1 Introduction

It is possible to make effective use of EMAS without any knowledge of its internal working, in the same way as it it is possible to be a good car driver without knowing how an internal combustion engine works. For those users who are interested in what goes on inside EMAS, this appendix provides a brief introduction.

A1.2 Virtual Machines

EMAS-3 shares the resources of the computer between the users in an equitable fashion, and in such a way that each user appears to have a computer to himself. This computer is made up of a virtual processor, a virtual memory of many hundred Mbyte, and file storage space on disc. The word "virtual" in this list of components can be read as "apparent" or "effective". The illusion implied by this terminology is clearly necessary, since the main store of the real computer - of the order of 16 Mbyte - is far less than that of even 1 virtual machine. Thus a very fast computer with a modest amount of main store and one processor is made to appear to be a large number of much slower machines, each with an enormous main store.

Further, while the user's virtual machine is of necessity much slower than the real computer, the user will not in general make constant demands on it: for example, sometimes he will be editing files (requiring relatively little computing), sometimes he will be compiling programs (requiring quite a lot of computing), sometimes he will be doing nothing at all. The System has to be able to cope with this variability without wasting resources.

A1.3 Processes

In sharing the resources of the machine, the System must work with a basic unit of scheduling. This unit is known as a *process*. It is an autonomous activity which competes with other processes for the resources available. Each user has a process. The purpose of EMAS can thus be described as "providing a suitable environment for each user process". In achieving this the System will have a variety of tasks to perform, some of them related directly to the users' requirements, some concerned with the housekeeping entailed in organizing the System as a whole. These tasks are also associated with processes and also compete for their share of the resources available.

Processes fall into two classes. First, "Supervisor processes", resident in main store, i.e. not paged. Secondly, "paged processes", which run in the virtual machines described above; these are created dynamically, for instance when a user logs in.

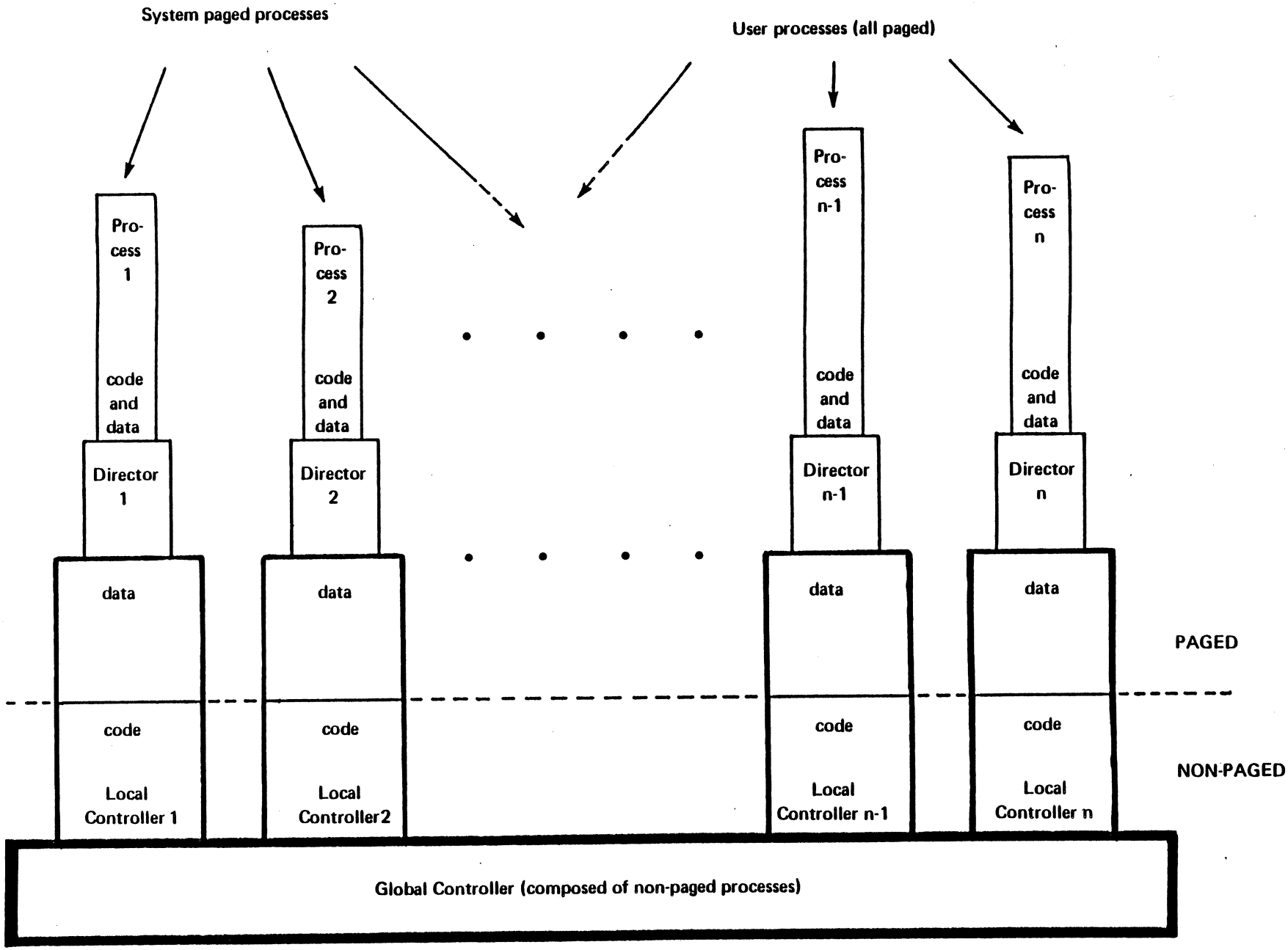


Figure 1.1: EMAS -3 System Structure

A1.3.1 Supervisor processes

The components of EMAS which are implemented as Supervisor processes are as follows:

Local Controllers

Each paged process has an associated Local Controller, whose main function is to manage the process's virtual processor and virtual memory. It ensures that the process stays within its current allocation of resources; if it requires more, the Local Controller will ask the relevant component of the Global Controller (see below) for more.

The Local Controllers are unusual in that their code is resident, while their data space is located in the virtual memories of their associated paged processes.

Global Controller

The Global Controller is central to EMAS, and is responsible for the following:

* Scheduling: Part of the Global Controller, known as the Scheduler, is concerned with sharing available main store and central processor time between the various Local Controllers competing for these resources. Since EMAS is primarily intended for interactive computing, the Scheduler is designed to give priority to processes which make comparatively small demands for store and processor time. Put another way, the Scheduler gives a lower priority to jobs which require large amounts of either resource. Thus processes which display characteristics of batch jobs, or are in fact started as batch jobs, only get resources when other processes are not waiting for them.

* Paging: Part of the Global Controller, known as the Paging Manager, is provided for Local Controllers to use. Its function is to bring pages into store as required. As this service is part of the Global Controller, it can take account of sharing of pages in main store. In other words, if a Local Controller has already requested a certain page, the next Local Controller to ask for it can be told that it is already in main store. This phenomenon occurs very frequently, particularly for editor and compiler pages, and helps to make good use of the available main store.

* Controlling peripheral devices: The Global Controller contains a number of device drivers, which control the disc drives and general peripherals (line printer, magnetic tape, etc.). There is also a device driver for the part of the hardware which links the main computer to the communications network. This link is used for interactive traffic (from users' terminals) and for file transfer to, for example, a line printer in another building.

A1.3.2 Paged processes

Each active user of EMAS has a paged process. The Supervisor processes described above provide each paged process in the System with a virtual machine, but not with software facilities. These facilities are provided by two components, which are both located in the virtual memory of the paged process in question (see Figure A1.2): the Director, and the Subsystem.

All the services provided for a user by EMAS are requested by calls on Director routines, which thus form a "procedural interface" between a user process and the rest of the System. These services, however, are at a basic level and are intended as the primitives from which a higher-level interface between the user and the System is to be built. Such an interface is known as a Subsystem, which in EMAS provides

such things as a command interpreter, text editors, compilers, a program loader, subroutine libraries, etc. The appearance of EMAS to a user is determined by the appearance of his Subsystem.

Most of this Guide is concerned with the facilities provided by the standard subsystem. However, as the Subsystem resides in his own virtual memory, it is quite feasible for a user to use a different subsystem, or indeed no subsystem, without affecting other users. Thus it is possible to have a number of different subsystems in use simultaneously in EMAS-3, each making use of the facilities provided by Director, but each appearing different to its users.

The Director, which is a collection of routines, maintains a table of information about the contents of the virtual memory; this is used to locate pages on the disc storage. Whilst it is running the Director can access the whole virtual memory, but when it hands over control to the Subsystem or to a user's program part of the virtual memory is made inaccessible, to prevent access to the Director's code and tables. This mechanism ensures that faults in the Subsystem or in user programs cannot result in corruption of the Director's tables. Since these tables include information about, for example, other users' files, this mechanism is essential to preserve the security of the System. Subsystem faults or user program faults can result in corruption of the user's own files and, at times, in his process terminating completely. What is important, however, is that no matter how serious the corruption in his own virtual memory, it cannot affect other users of the System.

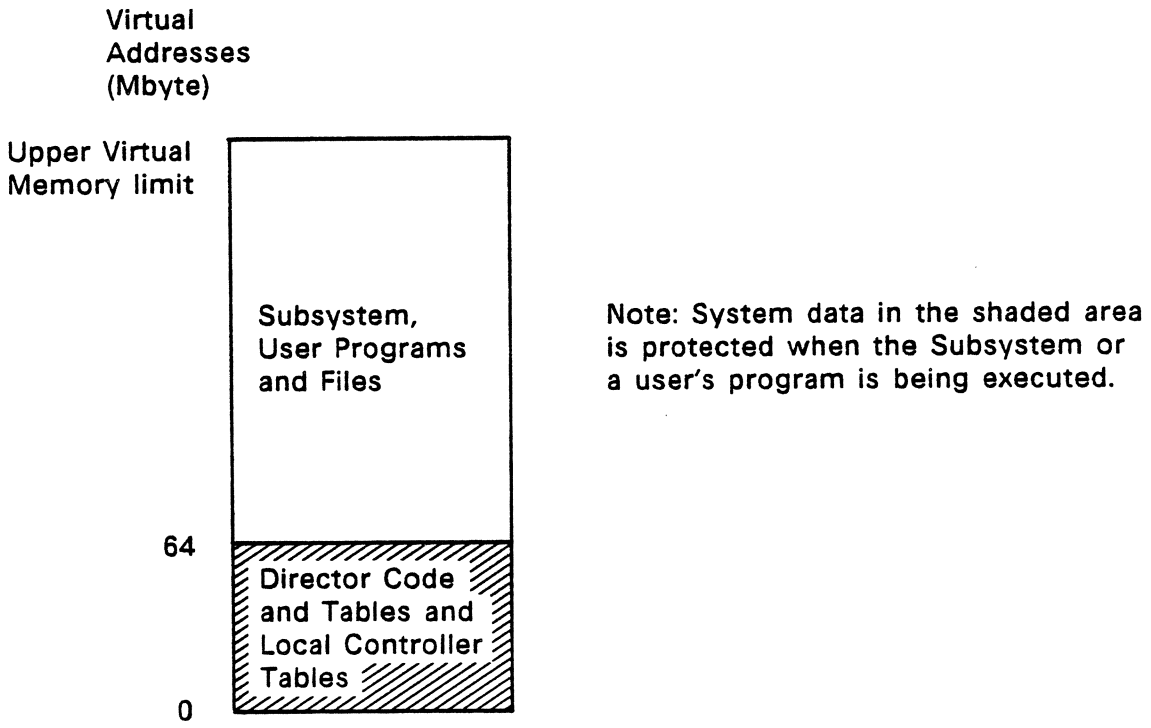


Figure A1.2 Layout of a Virtual Memory

Apart from maintaining the table of information about files connected in the virtual memory, the Director also performs the following tasks:

* organizing the disc file store (see below)

* communicating with System processes (see below)

* dealing with failures in the Subsystem or user program such as "CPU time exceeded" and "Zero divide".

A1.3.3 Paged System processes

Apart from the paged processes owned by users, there are a number of paged processes belonging to the System. In many ways they are similar to user processes but they have some special attributes:

* They are privileged - that is, they can access information and obtain services which are not available to user processes.

* They normally run without any interactive terminal.

The five paged System processes are the SPOOLR and FTRANS processes, the VOLUMS process, the DIRECT process and the MAILER process.

The SPOOLR and FTRANS processes are responsible for:

* handling the output of files to line printers, etc. (see Section 15.7)

* scheduling batch jobs (see Section 16.3)

* controlling communications with remote terminals including the transfer of files to and from other network hosts (see Section 15.10).

The VOLUMS process is mainly concerned with the organization of the backup and archive stores, described in Sections 3.8 and 3.9.

The DIRECT process is responsible for the verification of names and passwords at log on (see Section 2.3.3), and for carrying out consistency checks on the file system.

The MAILER process manages the reception, storage and delivery of electronic mail; communication between local users and users on remote hosts is supported. A distributed directory of recipient names for users of machines on the local network is also maintained.

APPENDIX 2
PAGING AND VIRTUAL MEMORY

A2.1 INTRODUCTION

This appendix contains a brief introduction to paging mechanisms and virtual memory. For two reasons it is felt necessary to explain these features of EMAS: first because they form a vital part of the System, and secondly because an understanding of them, although not necessary, can sometimes enable a user to make more efficient use of the System.

A2.2 STORAGE ALLOCATION

A2.2.1 The conventional solution

All computers have some form of storage for programs that are currently being executed and for the operands they access. This storage, referred to here as main store, has to be allocated in some way. In a simple single-programming computer all the main store can be allocated to one program (Figure A2.1). In a multi-programming computer main store has to be shared between more than one program, according to the requirements of the programs. Even when the number of programs being multi-programmed is small there are problems:

* Available programs may not conveniently fit into store; for example, if the store has 500 Kbyte available for user programs and all user programs currently awaiting execution require 200 Kbyte, then 100 Kbyte of store will be wasted.

* If programs have a wide variety of storage requirements the main store may quickly become *fragmented;* for example, if in Figure A2.2 program A terminates and the only available programs to run require 250 Kbyte, one will be loaded (program C) as in Figure A2.3, leaving a gap of 50 Kbyte at the top of the store. If program B now terminates there will be a total of 250 Kbyte of store free, but a single program requiring 250 Kbyte will not be able to run because the areas of store which are free are not contiguous.

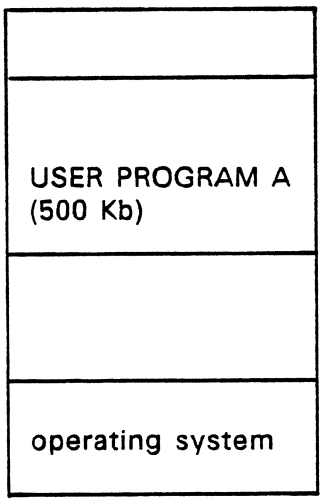


Figure A2.1

All available store allocated to one user program

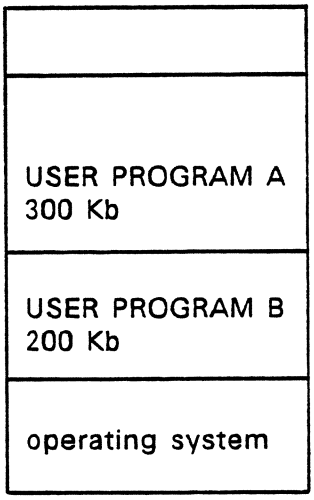


Figure A2.2

Available store allocated to two user programs

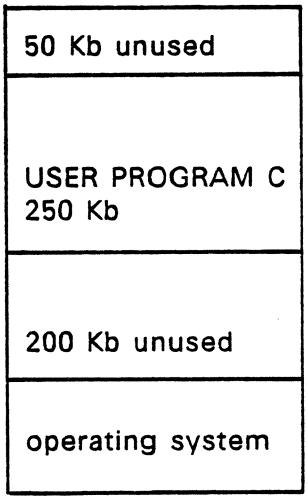


Figure A2.3

A second 250 Kb job cannot run because store is fragmented

A2.2.2 Further problems in a multi-access environment

Further problems of storage allocation arise in the case of a multi-access system:

* Whereas in a multi-programming batch system it is normal to allow typically three or four programs to share the available resources at any one time, in the case of a multi-access system 100 or more users may wish to use the computer simultaneously.

* The division of main store into discrete partitions of fixed size, which is feasible in a multi-programming batch service, is not practicable in a general purpose multi-access system where, for example, one user may be editing a program one minute, requiring very little store, and compiling it a few minutes later, requiring a large amount of store.

* As well as making varied demands for store each user will make varied demands for computing. Thus, whereas in a batch system each job normally runs to completion and is only delayed by waiting for peripheral transfers (for example blocks to be read from magnetic tape), in a multi-access system a user may take several minutes considering a reply to a prompt at his terminal. During this time no computing is being done for him and any store allocated to him is wasted.

Thus storage allocation in a multi-access system is concerned with many processes, each of varying size and making intermittent use of the central processor.

A2.3 PAGING

To allocate storage effectively EMAS uses paging. This involves some special hardware and part of the code in the Global Controller (see Section A1.3.1). The main store of an EMAS computer can be divided into pages of 4096 bytes each. Pages of store can be allocated to a program in any part of the store, wherever they happen to be available. The purpose of the paging mechanism is to modify the addresses of the pages so that to the running program they appear to be contiguous. Thus a user program might use four pages which it addresses as 0, 1, 2 and 3, although these pages might in fact be located at pages 8, 3, 14, and 0 in the main store (see Figure A2.4).

The name "virtual store" or "virtual memory" is given to the effective store addressed by the program.

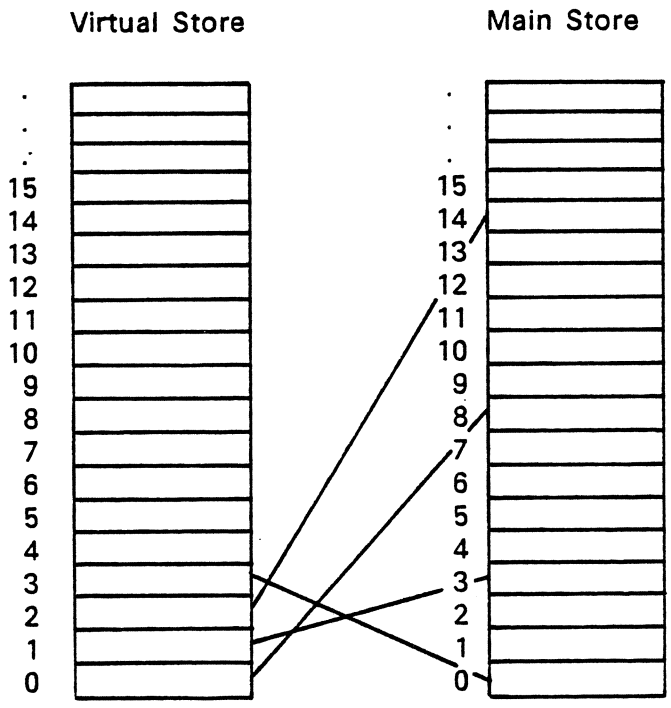


Figure A2.4 Example of paging

One of the problems not considered so far is the great difference between the total storage requirement of all active processes and the size of the main store. For example, for a mainframe with approximately 350 pages of main store available to user processes, during any period of a few minutes user processes may access more than 1000 different pages. The pages not in main store are held on backing store and are brought into main store as required. Each page used by a user process must have a location on the backing store to which it can be returned when its space in main store is required by another process. The backing store consists of one or more disc stores.

A2.3.1 Page faulting and virtual memory

An important characteristic of the paging technique is its ability to deal with the varying demands of a program for storage. If a program starts to access an array which it has not accessed for several seconds, the page containing the array may have been moved by the System from main store to backing store. As a result a "page fault" occurs and the program is held up while the Supervisor locates the page and reads it into main store. The program is then allowed to continue. The user's program knows nothing about this and need take no special action.

A2.3.2 The effect of paging on user programs

Paging enables the System to allocate storage efficiently between the users. In so doing it provides each user with a virtual memory, i.e. an addressable space, far larger than the real main store: a user of EMAS-3 can address an apparently continuous area of approximately 150 Mbyte.

One of the attributes of paging is that its operation is transparent to the user, who can proceed on the assumption that his "virtual" memory is real in every respect, and use the benefits of such an enormous main store. If one had such a main store, one could, for example, run far larger programs than a normal size of main store would allow, without having to resort to overlaying. More generally, one could dispense

with transferring information between backing store and main store in small sections, with all the complications of buffering entailed. Instead one could simply move all of the relevant files on backing store into main store and access them directly thereafter.

In EMAS this is precisely what happens: a file is accessed by first being "connected" into a virtual memory, at a suitable place, i.e. where there is an unused area big enough to take it. Thereafter reference to an address within the appropriate part of the virtual memory accesses the file directly. The System does not in fact move the complete file into some special area when connecting it; it merely notes the disc location of the file and *associates* that location with a range of addresses in the virtual memory in question. Connection is described further in Section 3.5. See also Figure A2.5.

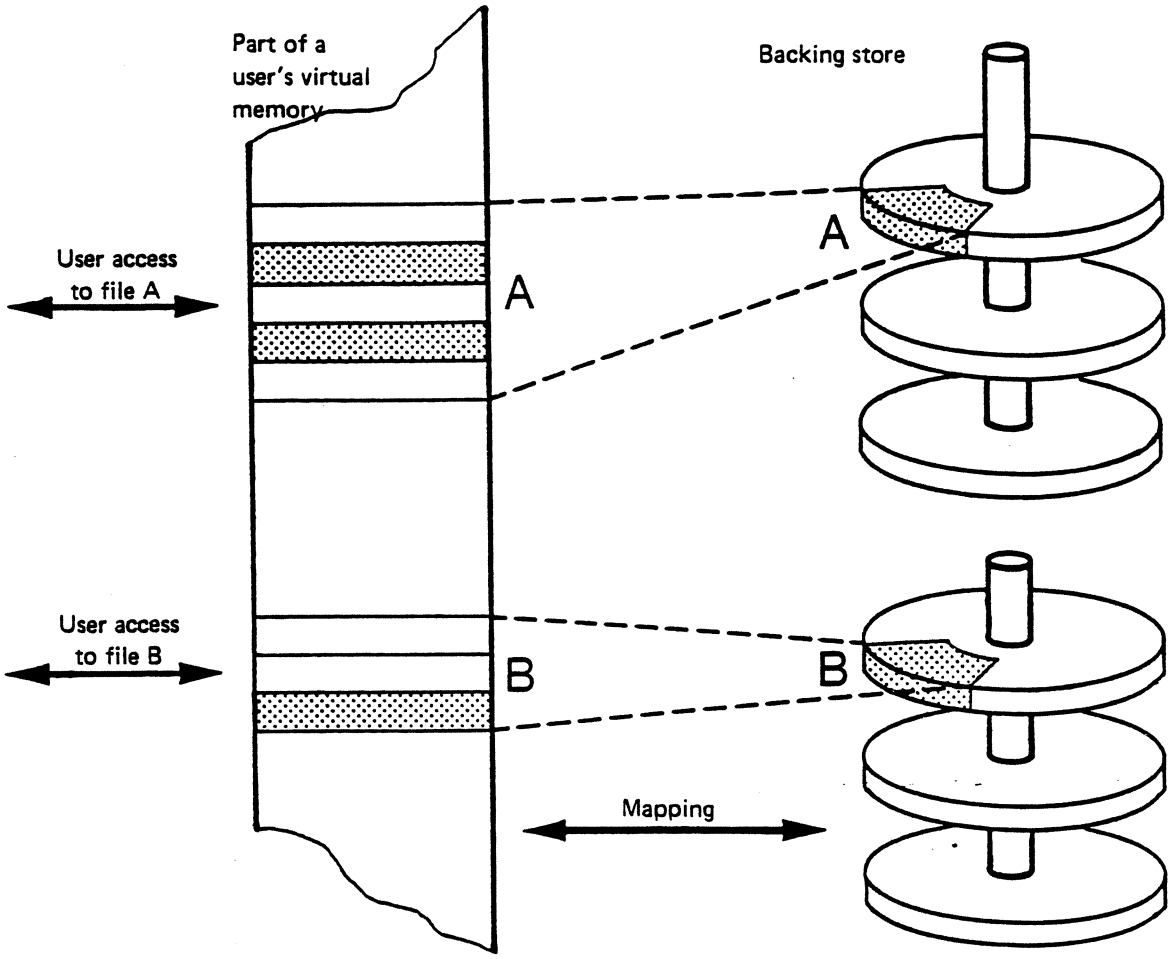
It is important to understand that there is no distinction, to the System, between data files and program files and even work areas such as the stack, used by IMP and Pascal programs to hold variables and arrays. Each connected file has an address in the virtual memory and consists of one or more pages, of which there is always a copy on disc and when the file is in use there will be a copy of some of its pages in the main store. An address in virtual memory is referred to as a "virtual address".

As explained in Chapters 8 and 9, file handling routines have been made available which simulate those normally provided, in order to simplify the transfer of programs to and from EMAS. In addition, however, there are routines available which enable users to map files onto arrays in their programs. These routines make possible efficient and powerful data manipulation with a minimum of code (see Chapter 19).

A2.3.3 Limiting page turns

In one important respect the use of a paged store is different from the use of a conventional store. Whereas in a conventional store variables can be accessed in a random order with no variation in the time required to access each one, in a paged store there will be a page fault each time a variable on a new page is accessed.

This page fault will not affect the results of the program but will affect the time taken to complete the program. In the case of programs which require a total of less than 1/4 Mbyte for the whole program and data area this is unlikely to be important. The programs that are affected are those which access, for example, elements of large matrices in a random order. Such programs are likely to have elapsed times far longer than programs which use equivalent amounts of central processor time but which do not access a large number of different pages. This is particularly relevant for large programs that are to be run in foreground mode, since the elapsed time is then the time the user will have to sit at a terminal awaiting the completion of his program.



Files A and B are connected into the user's virtual memory. The user accesses them as though they were part of contiguous main store - it is one of the functions of EMAS to sustain this fiction.

Each file is divided into fixed-size pages. A page of a file is only brought into "real" main store if it is referenced by a program or if code within that page is to be executed. The pages shaded in the diagram of the virtual memory represent those actually in main store; they will be copied to backing store, if they have been altered, when the main store space they occupy is required for other pages (probably by other users).

A file can be shared by two or more users, i.e. it can be connected simultaneously in two or more virtual memories. If a page of such a shared file is required simultaneously by two or more users, nonetheless only *one* copy of that page is held in main store.

Figure A2.5 File connection in EMAS

APPENDIX 3
CHARACTER CODES

EMAS uses internally the ISO character code. Within EMAS, code values in the range 0-127 are associated with graphical or control characters (see Table A3.1). The graphical representations of some codes vary from one output device to another; common alternatives are shown in the table.

The code values 128-255 are not associated with graphical or control characters within EMAS. However, codes which are in this range are not modified by EMAS in any way and can thus be used when sending character information to remote terminals or processors by means of a communications network.

The first value in each column is the decimal value of the character code.
The second is its hexadecimal value.

0	00	NUL	32	20	space	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	# (£)	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF*	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\ (¬)	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	↑ (^)	126	7E	~ (¯)
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

*Note that in EMAS, ISO code 10 (Line Feed) is normally known as Newline (NL). This is the character which is used as the line terminator in character files (see Section 13.3) and in character input and output.

Table A3.1 EMAS Internal Character Code

APPENDIX 4
ERROR MESSAGES

4.1 Error messages at log-on

The following error messages may be produced when you attempt to log on to EMAS.

`Invalid Name`

The username supplied in the log-on sequence does not consist of six printable characters.

`Invalid pass`

The password supplied is not correct.

`Process Running`

Normally a maximum of one foreground process is allowed for a particular username at one time (the number is alterable at the discretion of the System Manager). If the maximum allowable number of processes for the username are already extant then this is the failure message.

`No User Service`

The service is not available due to software or hardware testing. Log on to the INFO host to discover the reason.

If you get some other message when you log on, which you do not understand, then you should contact the Advisory service.

4.2 Subsystem Error Messages

Common error messages produced by the Subsystem should be self-explanatory. Some of the general ones are described more fully below; a list of Subsystem error messages may be found in the file SUBSYS:ERRORTEXTS.

4.2.1 General error messages relating to files

These messages can be produced following faults in many commands. The character "&" in a message is replaced by the relevant name at the time of the failure.

`Conflicting use of file & in another process`

This message will be produced when an attempt is made to connect a file in a way which is incompatible with the use currently being made of it by another user. For example:

When an attempt is made to read from a file which belongs to another user (and is permitted to this user in READ mode) at the same time as the owner of the file is running a program which is writing to the file.

`File connected in another VM`

The file is currently connected in another virtual memory. While this is the case it is not possible to DESTROY, RENAME or OFFER it.

`File & does not exist`

This message is produced when a user attempts to access a file belonging to himself which does not exist.

`File & does not exist or no access`

For reasons of security when accessing files belonging to other users, no distinction is made between the fact that a file does not exist and the fact that it is not permitted to this user.

`File & in use at a lower level`

This message indicates that you are attempting to do something to a file which is being used, by you, already. For example, if you try to DESTROY the file you are currently OBEYing.

`No free descriptors in file index`

This message indicates that there is insufficient room in the user's file index to hold information about the file being created. This can arise unexpectedly because some commands require the Subsystem to create files which the user does not know about. For example, `LIST FILE27,.LP` makes a temporary copy of `FILE27`.

`No more space for output`

This message will occur if the file being created has reached its maximum size, or the user's total file limit has been reached or the disc holding the user's files becomes full.

`Inconsistent file use`

This message will occur when an attempt is made to operate on a file in a way which is inconsistent with its current use. For example, the command

```
Command: IMP TEST,TEST
```

attempts to compile into a file from which source is being read.

`Invalid filename &`

The rules for naming files are contained in Section 3.2.2.

`Invalid username &`

This message will occur if an attempt is made to access a file belonging to a user whose files are not available. This may be because no such username exists, or because the disc containing the user's files is not currently on-line.

`Requested access permission not allowed`

If a file is permitted to this user in READ mode and an attempt is made to connect the file for writing, e.g. `EDIT filename`, this message will be produced.

4.2.2 Other Subsystem failure messages

It is hoped that other messages produced by the Subsystem will be self-explanatory when read in context. The production of clear diagnostic information both in relation to the Subsystem and to user program failures is regarded as being of importance. Further improvements are being made in this area. Users are encouraged to contact the Advisory service if they have problems relating to the error messages produced by the Subsystem.

4.2.3 The source of failure messages

The Subsystem allows programs and commands to call other commands. This can give rise to failure messages which appear to be out of place. If, for example, you get a failure message such as:

```
COPY fails - Existing file ABC has different type
```

when you have not been using the COPY command then this will probably be because you have called a command, or run a program, which itself calls COPY.

APPENDIX 5
ENVIRONMENTAL QUERIES

Obtaining information about the Subsystem

There is sometimes a requirement for a program to determine information about the environment in which it is running. There are two entries, EMAS3UINFI and EMAS3UINFS, which can be called by your program to obtain this information.

The routine EMAS3UINFI returns an integer value. The routine EMAS3UINFS is used for textual items and returns a character string value. Note that both have been written to be callable from several languages. Sections 8.6.4 and 9.4.7, 9.4.8 explain how to call them from programs written in IMP and Fortran respectively.

The result obtained by calling any of these subprograms is a single item of information, specified by the value of ENTRY. The values of ENTRY which can be specified with these functions, and the meanings of the results obtained, are given in Table A5.1: for any other value of ENTRY, the value 0 or a null string (spaces in Fortran) is returned. Further ENTRY values may be defined later, but the meanings given below will not be changed.

ENTRY	EMAS3UINFI result
1	own FSYS (logical disc) number.
2	Mode - foreground=1: batch=2: foreground OBEYfile, 3.
3	Number of users.
5	Current CPU limit in seconds.
6	Max file size in Kbytes.
7	SYNC1DEST
8	SYNC2DEST
9	ASYNCDEST
10	ADIRINF
11	PROCNO
12	FDLEVEL
13	Incarnation (as an integer) to distinguish multiple simultaneous processes for the same user number.
15	ITWIDTH (interactive terminal width).
19	Returns 1 if resources are are scarce - else zero.
20	FUNDS left, in pence.
21	Charge for this session, in pence.
22	Returns non-zero if messages are inhibited.
23	Terminal type (as set by command TERMINALTYPE).
24	Front-end comms. stream identifier (1).
25	Front-end comms. stream identifier (2).
27	User Stack Size (Kbytes)
28	Initial Work File Size (Kbytes)

ENTRY	EMAS3UINFS result
1	username for current process.
2	"DELIVER" text.
3	Process start time.
4	Current prompt text.
5	Current active directory name.
6	Subsystem version.
7	Surname.
9	Name of FUNDS holder.
10	OCP type.
11	Batch job name.
12	Supervisor version.
13	Director version.
14	Terminal address. N.B. some characters may be non-printable.
15	Network name
16	Service name.
17	Current group.

Table A5.1 EMAS3UINFI and EMAS3UINFS Summary

Notes

The data returned by EMAS3UINFI for ENTRY 7 to 12 are not of general interest, and if you do not recognize their somewhat cryptic names, then you probably do not need to use them.

The values returned by EMAS3UINFI ENTRY 24 and 25 are concerned with communications software. They have nothing to do with currently selected input or output streams.

The values returned by EMAS3UINFI ENTRY 4, 14, 16, 17, 18, 26 and EMAS3UINFS ENTRY 8 have no defined meaning.

APPENDIX 6
GLOSSARY

This appendix gives brief definitions of a number of terms and abbreviations used in the Guide and elsewhere in relation to EMAS-3. It should be appreciated that some of the terms do not have generally accepted definitions and so may be used for different purposes with respect to other systems.

Access Permission	An attribute of a file or file group, which can have several associated access permissions. Each one specifies a user, a group of users, or all users, and a corresponding permitted level of access. Access permissions determine which connect modes are possible.
Accredit	To register a user on the EMAS System. This is done by the System Manager, and entails giving the user a Process Index and access to it by means of a Username and password.
Acoustic Coupler	A hardware device which can be used in some cases instead of a Modem. It does not require the special wiring required for a modem. It allows you to connect an interactive terminal to EMAS via a standard telephone.
Address	An integer value which uniquely specifies a byte location in a store (memory). A "physical address" refers to a location in the computer's main store; a "virtual address" refers to a location in a virtual memory. Addresses of data items which are larger than a byte specify the leftmost byte of the data items.
Archive Index	Each user has an index of those of his files which are held in the archive store (cf. File Index, below).
Archive Store	A long-term store for users' files, held on magnetic tape (see Section 3.9).
Background Mode	A mode of use of EMAS where access is not from an interactive terminal. Background jobs may be put in a queue by the user during a foreground session for execution. They can also be submitted from other computer systems.
Backup Store	A magnetic tape based store holding copies of users' disc files, the purpose being to enable their restoration should they be corrupted or destroyed through hardware or software malfunction. This is distinct from archiving, in which disc files are copied to magnetic tape and the disc copy then deleted.
Byte	The smallest addressable unit in the main store (or virtual store) of computers used for EMAS. It contains 8 binary bits. It is used in IMP for variables of type %BYTEINTEGER and in Fortran for variables of type CHARACTER*1.
Connection	The only method provided by EMAS for gaining access to files. It is a simple concept central to the provision to each user of a virtual memory: see Appendix 2.
Console	Alternative name for an Interactive Terminal.

CPU	(Stands for "Central Processor Unit".) The piece of hardware central to the computer which coordinates and controls the activities of all the other parts and performs the arithmetic and logical operations on data.
DIRECT	A paged System process which controls the logging on of users (checking names and passwords, initiating user processes, etc.). It also carries out consistency checks on the File Indexes at IPL time.
Director	A collection of routines, forming part of the EMAS System. Each user process has its own Director. If EMAS is seen as an onion-like structure, Director is the level directly below the Subsystem. All the services provided for a user by EMAS are requested by calls on Director routines, which thus form a procedural interface between a user process and the rest of EMAS. Normally, however, there is another layer between this interface and the user, viz. the Subsystem. Director is paged; it is contained in the associated user's virtual memory (See also Appendix 1).
Disc Storage Unit	A magnetic storage device consisting of a number of flat circular plates, each coated on both surfaces with some magnetizable material. The plates are mounted on a central column which is rotated at high speed. On each surface there is a series of concentric tracks. A read/write head (one per surface) can be moved radially to a required track for accessing or changing the information stored there. When used on EMAS-3 each unit is treated as four logical units. Each of these logical units contains a File System (cf).
Disc File	See File (below).
Disc Store	See Store (below).
EDNET	Edinburgh Regional Computing Centre Communications Network. This is the combination of computer hardware, software and cabling which connects the mainframe computers operated by ERCC, and those operated by many University Departments and Research Institutes. It provides access to all these computers from more than 1000 interactive terminals, as well as having links to other networks.
File	A collection of information held within some storage device in the computer system. EMAS operates - from the user's point of view at least - in terms of files, and extensive facilities are provided by Director for using them, keeping them, sharing them, etc. Director does not impose any conditions or conventions regarding the interpretation or use of the contents of files. The Subsystem provides facilities for working with particular types of file (see Chapter 4). A user's disc files are all kept on a single logical disc unit.
File Group	(in some contexts just 'group'). On-line files can, optionally, be collected in groups. A file group can contain zero or more files and zero or more other groups. It has a local name which consists of up to 11 letters or digits, of which the first must be a letter.

File Header — An area at the beginning of a file that contains information about the type, size and, in the case of data files, format of the file.

File Index — Each accredited user of EMAS has a File Index which is maintained by his Director. It keeps information about all the disc files belonging to that user, including their sizes, access permissions to other users, etc. The File Index is kept on the same logical disc unit as all that user's files.

Filename — (Sometimes written as two words.) The name given to a file by a user when it is created or renamed. The name must be unique within that user's File Index, but other users may have files of the same name; see also Full Filename. For primary files the name consists of up to 11 letters and digits of which the first must be a letter. For files which are members of groups the name will take the form {groupname.}localname, where the part in curly brackets will appear one or more times.

File System — Each user of the System has a File Index, which contains information about all his disc files. Each user's File Index and associated files are held on a single logical disc unit.
A file system consists of the files and File Indexes held on a single logical disc unit, together with other information stored there to manage the space allocation and to ensure the self-consistency and integrity of the logical disc unit.
The file system is the collection of all the logical disc unit file systems. Note that the term "File System" refers only to on-line files, i.e. those on disc.

Foreground Mode — A mode of use of EMAS where access is from an interactive terminal.

FTRANS — A paged System process which controls the transfer of files between the computer running EMAS and other computer systems. This includes MAIL messages between EMAS users and users on these remote systems.

Front End Processor — (Sometimes called "FEP".) A computer which is directly connected to the computer running EMAS. Its job is to manage communications with remote processors and terminals. A large EMAS mainframe may have several Front End Processors.

Full Filename — A filename together with the owner's username. The two parts of the full filename are joined by a ':' to form a single string: thus "username:filename". The full filename is unique to that file and must be specified by users other than the owner when referring to the file.

Group — See File Group

Hardware — The physical components of a computer, such as the central processor unit, the magnetic tape units, etc.

Head Crash — A hardware fault usually affecting a single logical disc unit and corrupting or destroying some users' disc files. It occurs when a recording head comes into contact with the surface of a disc, causing physical damage to the recording medium.

Hexadecimal

A number system using a radix of 16. The digits are represented by the characters 0-9 and A-F. It is a convenient system to use on an 8 bit byte machine, since the contents of each byte can be represented by two hexadecimal digits.

IMP

A block-structured problem-oriented high level programming language of the ALGOL type. It was developed in Edinburgh from Manchester University's Atlas Autocode, primarily for implementing system software - hence its name. Apart from a very small amount of machine code, the EMAS System software is written entirely in IMP; this includes the Subsystem and the Fortran77 and IMP80 compilers.

Interactive

Descriptive of the type of computer use which permits a user to respond to the output as it is being generated, and thus to stop or change the course of a computation. For such interaction a user console is required. With the Edinburgh Subsystem of EMAS three levels of interaction are permitted to a user:

1) after each command has been completed, successfully or unsuccessfully, the user is asked for his next command;
2) during the execution of a program, the user may be asked for input data, depending on how the program was written;
3) the user can interrupt any program running in his process and redirect or abort it. The standard alternative to interactive (or foreground) operation is batch (or background) operation.

Interactive Terminal

The general name for a terminal used for accessing EMAS in foreground mode - i.e. interactively. It can be a hard copy terminal, such as a Teletype, a VDU or a microcomputer running a suitable terminal emulation program.

Also called a console.

Interrupt

A signal generated either externally, for example by an interactive terminal, or from a running program which normally causes the current activity to be suspended until the contingency implied by the interrupt is dealt with.

I/O

(Stands for "Input/Output".) Information travelling to or from a computer system.

I/O Device

A piece of hardware, attached to a computer, which can read, store or write information in some form.

IPL

(Stands for "Initial Program Loading".) Strictly a hardware function involving reading in a minimal control program to an otherwise empty machine. In the case of EMAS the term covers the whole process of initialising the computer for an EMAS session. This entails initialising store and the peripheral controllers, carrying out a consistency check of the File Indexes, and starting the continuously running paged System processes.

Localname

Each file and file group has a localname which is the final element of its full name. In the case of a primary file or group the localname distinguishes it from other primary files and

groups belonging to the same user. In the case of a file or a group which is a members of a group the localname distinguishes it from other members of the same group. The localname consists of up to 11 letters or digits of which the first must be a letter.

MAILER — A paged System process which manages the reception, storage and delivery of electronic mail. It supports communications between users of EMAS and users on remote hosts.

Main Store — The part of the computer hardware which can store instructions or data to be executed or read or written. With respect to the EMAS System, main store is divided into "page frames", each able to hold a Page. It is allocated thus to user processes, although EMAS manages main store and backing store in such a way that they appear to form a continuous main store much larger than the pages of main store actually allocated. See also Store, below.

MANAGR — A paged System process administered by the System Manager, and used for accrediting users, changing control parameters for users, etc.

Memory — A synonym for Store.

Modem — A hardware device connected between an interactive terminal and a telephone circuit to convert signals from the terminal into a form suitable for transmission over the telephone circuit, and vice versa.

On-line —
1) Of peripheral equipment: logically connected to the EMAS System, the alternative being "off-line".
2) Of a user: having successfully carried out the standard log on procedure via an interactive terminal, so that the associated user process has been started up and interactive working can be undertaken.

Operating System — A suite of programs that controls usage of the computer; in EMAS this includes the Supervisor, Director, Subsystem and the paged System Processes.

PAD — (Package Assembler Disassembler.) This is a small computer which connects a number of interactive terminals to a communications network. For the terminals to be able to access EMAS the Network must be connected, directly or indirectly, to the EMAS computer.
Data is sent around the network in packets (blocks of typically 50-100 characters). For input the PAD reads individual characters from the terminal and assembles them into packets. For output it disassembles the packets received from the network and sends them, one at a time, to the terminal.

Page — The basic unit of store, and of file space allocation in EMAS. A page contains 4096 bytes (4 Kilobytes). Main store is divided into pages, and information is transferred between main store and backing store in units of pages. From the point of view of a user or a Subsystem, however, information is handled in terms of files, the paging operating transparently. See also Units of Storage, below.

Paged System Process

Similar to a normal user process but belonging to the System. The paged System processes are used to carry out System tasks such as spooling files, managing the archive store, running Engineer test programs, etc. They differ from user processes in that:

1) they may have access to facilities not given to users;
2) in some cases they are not run from an interactive terminal;
3) in some cases they may not make use of the standard Subsystem.

Page Turn

A term used to describe the transfer of a page to or from main store. Page turns are recorded, and the total number of page turns caused by a process is used as accounting information.

Paging

A method of accessing main store by dividing it into page units and addressing them through a translation mechanism. When used in conjunction with backing store of discs, it allows many processes with varying storage requirements to use a limited main store as if each had a large allocation of continuous store. See Appendix 1.

Process

An autonomous activity which can compete for the resources of the System. It is the basic unit of scheduling. The main function of EMAS is to provide a suitable environment for user processes. Each user process or paged System process is autonomous in the sense that it apparently has a virtual processor and virtual memory to itself. In a single-processor computer only one process will actually be executing at any instant, but the resources are shared in such a way that all active processes appear to be executing.
Two types of process are distinguished: a virtual process, which is paged, and a Supervisor process, which is resident in main store. See Appendix 1.

Process Index

A user's index which contains information that is required to be retained between sessions, such as his passwords and delivery information.

Program Loading

The process of preparing for an object file to be executed, which entails connecting the file in virtual memory and satisfying any external references it makes. See Section 7.5.

Route

A source or destination for data for a program or command. A route has an identifier (either an integer or a character sequence). It also has a specification. This includes the name of the file(s) or device to be used, as well as details such as the maximum size of file to be created and the format of records to be written.

Segment

The virtual memory is divided into segments of (on EMAS-3) 1 Mbyte. Each file in use by a process has to be connected at the start of a segment. This is of little significance to user programs except that it imposes an absolute limit on the number of different files which can be connected at one time in your virtual memory (approximately 200). Note that a file may be longer than one segment, in which case it will need more than one segment in the virtual memory when it is connected. It will always need at least one segment, even if it

is very short.

Software — A general name for programs as distinct from hardware. Also used to describe components of the operating system, as against user and applications programs.

SPOOLR — A paged System process which controls the transfer of files between the computer running EMAS and

1) local slow peripherals, or
2) remote devices (via the Front End Processor). All input files destined for, and output files generated by, user processes are transferred via SPOOLR, which also handles and schedules batch jobs, however submitted.

Stack — An area of store for each process (see above) used on a last-in first-out basis. The stack holds information about procedure calls, work variables used in the evaluation of expressions, and program variables which only exist for the duration of a procedure call e.g. normal IMP variables and arrays.

Store — Three types of physical storage media are distinguished in EMAS: archive store (magnetic tape), disc store, and main store. A file is either archived (held on magnetic tape) or it is a disc file.

Subsystem — The facilities provided by EMAS are made available to users via the Director procedural interface. These facilities, however, are at a basic level and are intended as the primitives from which a higher-level user interface is to be constructed. Such a user interface is known as a Subsystem, which in EMAS is intended to provide such things as text editors, compilers, program loaders, subroutine libraries, run-time support, etc. Each user has his own Subsystem, although in practice this is almost always an incarnation of a standard Subsystem. This Guide describes the facilities provided by the standard Subsystem.

Supervisor — That part of the EMAS System below the Director. See Appendix 1.

Supervisor Process — See Process, above.

System — "Computer System" is the term used to describe the composite of a computer and associated software; it may also include satellite computers (e.g. Front End Processors) and associated software.
The "EMAS-3 System" (or "EMAS System"), is sometimes used to mean the software up to the Director interface, plus the paged System processes. More usually it implies a computer with EMAS software, up to and including Subsystems and user programs.

System Manager — The person who is responsible for the day-to-day running of the EMAS System, and who takes the management decisions which arise (such as those relating to the accrediting of users).

TCP — Terminal Control Processor. Serves the same function as a PAD (see above). TCPs are being replaced by PADs throughout EDNET.

Terminal — A hardware device used for communicating with a computer. It may be an interactive terminal, such as a VDU, or a remote computer used for line printer output.

Track — On a disc, the part of the surface which is accessible from a single read/write head position.

Units of Storage — The table below shows the relationship between the main units used for measuring storage:

		Bytes	Kbyte	Pages
1 Mbyte	=	1048576	1024	256
1 Page	=	4096	4	1
1 Kbyte	=	1024		

User Group — Files can be made accessible to individual users, or to groups of users. A User Group is represented by a six character field containing one or more question marks. For example EGNP?? represents the User Group which contains all users whose usernames start with 'EGNP'.

User Process — See Process, above.

Username — A unique six-character alphanumeric name assigned by the System Manager to a user when he is accredited, and used to distinguish him from other users. The user's process is given this name, and the full filename of each of his files includes this name. The user must specify it when he is logging on to EMAS.

Virtual — The primary function of the System is to maintain for each user process the illusion that the process has a complete machine at its disposal without interference from other users. Most of the features of EMAS follow logically from this objective. The components of this illusion are described as "virtual", to distinguish them from their "real" counterparts in the actual computer.

Virtual Address — An address by which a program accesses a location in a virtual memory. The virtual address is converted by the paging mechanism into a real address in the main store. The IMP function ADDR returns the virtual address of a variable. See also Appendix 2.

Virtual Memory (VM) — An apparently continuous, addressable area of about 200 Megabytes. It is mapped by the paging mechanism onto pages in the main store, or on disc. Of the 200 Megabytes in a complete virtual memory, about 150 Megabytes are available to the Subsystem and user programs and files.

Visual Display Unit (VDU) — An interactive terminal which displays its output on a cathode ray tube rather than using paper.

VOLUMS — A paged Systems process responsible for the backing up, archiving and restoring from archive of users' files. See Sections 3.8 and 3.9.

BIBLIOGRAPHY

The following documents and books provide further information. All are available from the ERCC Advisory service.

EMAS 2900: Concepts - P.D. Stephens. ERCC. 1980. (Most of the information in this applies equally to EMAS-3.)

Edinburgh FORTRAN77 Language Manual - Editor N. Hamilton-Smith. ERCC. 1985.

The IMP80 Language - Felicity Stephens and John Murison. ERCC. 1982.

Electronic MAIL on EMAS - S. Shaw. ERCC. 1986

Apart from these major manuals there is a vast amount of useful information for users of EMAS-3 in the User Notes published by ERCC. These are available, free of charge, from the ERCC Advisory service. Several of these are referenced in the body of this Guide. The most important is User Note 0. This contains a list of all the other Notes and an index to them all. This User Note is updated regularly to reflect the changes and developments in the other Notes.

INDEX

This index gives references in the form Chapter.Section.

! 17.3,17.3.1,17.3.3
!= 17.3.2
& 3.4.4,3.5.1,3.9.4,7.5.6,7.6.3,7.6.4,16.3
* 3.3,3.9.2,3.9.5,4.3,14.10,16.3,17.2,21.4
** 3.3
+ 4.4,4.5,5.6.4,7.2.1,13.6.2,15.7.1,18.6
, 2.5.1
- 3.3
. 3.2.2,3.2.3,3.3,3.9,15.6.2
: 3.2.1-3.2.4,A6
= 3.2.4,7.2.2,7.2.3,7.5.3,7.5.4,7.5.7,17.7,17.9,18.6.1
? 3.3,3.7.1,3.9.5,3.9.6,15.10,15.11.1,15.11.3,17.2,17.9,18.5.1,18.5.3,18.5.4,18.5.6
?? 17.2,18.5.1,18.5.4,18.12
@ 17.2,21.5.3,21.5.4
\ 3.3,3.9.5
_ 3.3

ACCEPT 3.6.2,3.6,18.5.4,20.2
accredit A6
accredited user 2.2.1,3.2
acoustic coupler A6
active directory 7.6.2,7.5.3,7.6.3-7.6.5,8.2.1,17.6,A5
ACTIVEDIR 7.6.2,7.5.5
address A6
  physical A6
Advisory service 2.3.2,2.7.1,2.8,5.1,6.4,6.5,6.7,12.3,14.4,15.2,15.7.1,15.12,21.2,21.5,21.6,21.7, A4.1,A4.2.2,Bib
AFTER (DETACH parameter) 16.3
ALERT 2.7.5,2.3.3,2.7.2,2.7.6
ALIAS 17.6,7.6,7.6.2,17.1
.ALL 3.3,3.7.1,3.7.4,3.9.6,18.6.1,21.4.2
ampersand 3.4.4,3.5.1,7.5.6,7.6.3,7.6.4,16.3,18.6.1
ampersanding 17.2
ANALYSE 4.3,3.4.1,3.5,3.7.1,7.3.1,7.7,9.2,13.2,17.2,17.6,20.6
APERMIT 3.9.6,3.7,3.9.3
ARCHIVE 3.9.1,3.9
archive index 3.9.3,A6
archive store 3.9,3.2,3.7,3.9.1,3.9.3,3.9.5,14.8,A1.3.3,A6
  files 3.9.2
ARRAYDIAG (OPTION keyword) 17.9
asterisk 3.3,3.9.2,3.9.5,4.3,14.10,16.3,17.2,21.4
ATTR (PARM option) 9.2
AWAITRESTORES 3.9.4

background mode 8.7.2,9.5.6,16.1,16.2,A6
backup store 3.9,14.8,A1.3.3,A6
basefile 7.5.3
BATCH 16.4,15.9.1,16.1,16.3,16.5
batch job 3.9.4,9.4.3,9.4.4,13.6.2,15.9.1,16.6,17.4,17.9,A1.3.1,A6
  cputime limit 16.3
  deleting 16.5
  journal 16.3
    size 16.3
  ordering 16.3
  priority 16.3

queue 16.1,16.3,16.4,17.4
entry number 16.4,16.5
scheduling A1.3.3
time delay 16.3
binary file 12.3
BLANKLINES (OPTION keyword) 17.9
BSTARTFILE (OPTION keyword) 17.9
byte A6

C# 7.5.5,18.2,18.11,18.12
C programming language 11
card, punched 15.2
carriage control character 9.5.3,9.5.4
carriage return 13.3
CAST 6.3,6.5
central processor unit A6
CHANGESM (IMP routine) 19.4.6
character code
EBCDIC 21.4.3
ISO 21.4.3,A3
character file 13.3,2.5.1,3.3,3.4.1,3.5,4.2,4.4,4.5,5.3,5.11,7.2.1,8.7.1,8.7.2,9.5.1,9.5.3,13.2, 16.2,16.3,17.2,17.4,18.4,18.5.4,18.6.1,18.7.1,19.3,19.4.5,20.2,21.5.3
as output 9.5.4
creating 4.7,5.2,5.3,5.6.4,5.11,5.18,8.7.2,16.3
listing 15.7.1
overwriting 2.7.6,3.3,4.7,5.3,5.6.4,5.11,5.18,8.7.2,16.3
character input/output 8.7.2,8.7
charge 17.8,A5
CHERISH 3.8.1,20.2,20.3
CLEAR 13.9,9.5.2
CLOSESM (IMP routine) 19.4.4,19.4.6
code
executable 7.3.1
executing 7.4
object 9.2
renaming 7.3.3
shared 7.3.1
unshared 7.3.1
CODE (PARM option) 8.2,9.2
colon 3.2.1-3.2.4,A6
comma 2.5.1,15.7.3,17.3.3,18.7.1
command 2.4,2.9,4.1,8.2.1,16.2,A4.2.3,A6
abandoning 2.5.3
ACCEPT 3.6.2,3.6,18.5.4,20.2
ACTIVEDIR 7.6.2,7.5.5
ALERT 2.7.5,2.3.3,2.7.2,2.7.6
ALIAS 17.6,7.6,7.6.2,17.1
alias 4.3
alias name 7.6
ANALYSE 4.3,3.4.1,3.5,3.7.1,7.7,13.2,17.2,17.6,20.6
APERMIT 3.9.6,3.7,3.9.3
ARCHIVE 3.9.1,3.9
AWAITRESTORES 3.9.4
BATCH 16.4,15.9.1,16.1,16.3,16.5
calling 17.2
from FORTRAN 9.4.5
from IMP 8.5
CHERISH 3.8.1,20.2,20.3
CLEAR 13.9,9.5.2
CONCAT 4.5

CONVERT 4.7,4.5
COPY 4.4,2.5.1,3.2.4,3.4.1,3.7.1,4.5,20.5,20.8,A4.2.3
COPYTAPE 21.5.4
CPULIMIT 17.7,7.4.1,17.1
DATASPACE 19.5.2,19.5.1,19.5.3-19.5.5,19.5.7
DEFINE 13.6,7.4.1,8.5,8.7.1,8.7.2,9.5.2,9.5.5-9.5.7,13.8,13.9,13.10,17.9,18.7.1, 19.3,19.4.1,19.4.5,19.4.6,20.5,20.6
DEFINEMT 21.4,13.7,21.4.1,21.5.3,21.6
DELETEDOC 15.9.1,15.7.1,15.9,15.11.3,16.5
DELIVER 15.7.3,16.3
text A5
DESTROY 3.4.4,3.5,3.7.1,20.7,A4.2.1
DESTROYGROUP 3.4.5
DESTROYSET 3.4.6
DETACH 16.3,2.5.2,4.2,4.5,9.5.6,16.1,16.4,16.5,17.4
DISCARD 3.9.5
DISCONNECT 3.5.1,3.5
DOCUMENTS 15.6.1,15.8,15.8.1
DPLIST 15.7
ECCE 5.11,5.1,8.9.2,13.3,14.10,16.2,17.4,20.5,20.8
echo 17.9
EDIT 5.3,2.5.1,3.4.1,3.7.1,5.1,5.2,13.3,14.10,16.2,17.3.3,17.4,20.8,A4.2.1
entry name 7.5.5
executing series 17.4
execution 7.5
file 16.6,17.4,17.5
FILES 3.3,2.4,3.4.6,3.9.2,13.3
FORTRAN 7.2.1,2.5.1,3.9.4,4.5
FROMTAPE 21.5.2,21.5.3,21.6
GPLIST 15.7
GROUPTOPD 20.9.1
HAZARD 3.8.1
HELP 2.7.4,2.7.2,2.7.6,6.5,14.4
help information 18.5.4,18.12
history mechanism 17.3,17.1
HOLDMT 21.4.2
HOSTS 15.10.2
IMP 7.2.1,3.4.1,4.5,17.3.1,20.6,A4.2.1
INITPARM 7.2.3,7.1
INSERT 7.6.3,7.6.2,7.6.5,8.2.1,17.2
interpreter 7.6.3,A1.3.2
line 18.3
case 18.10
continuation 2.5.1,17.4
manipulation 18.10
setting 18.10
spaces 18.10
stored 17.3.1,17.3.2,17.3.4
LINK 7.3.2,4.6,7.1
LIST 15.7.1,3.5,4.5,9.5.4,15.7,15.8,15.8.1,15.9,20.6,A4.2.1
LOADPARM 7.5.7,7.5.2
LOOK 5.8,5.1,20.8
LOOKFOR 7.5.5
macro scheme 17.5,17.1
MAIL 15.12,2.3.6,14.8,14.10
MAKECOMMAND 18.11
MESSAGES 14.9,14.10
METER 17.8,2.4,2.7.6,7.5.5,17.1,17.3
MODIFY 7.3.3,7.1,7.7

name 2.4,1.5.6,2.5,2.5.4,7.5.5,7.6,8.5
abbreviation 17.6
NDELETEDOC 15.11.3
NDOCUMENTS 15.11.2,15.8,15.11.3
NEWDIRECTORY 7.6.1,3.4.1
NEWGEN 3.4.3,3.7.1
NEWGROUP 3.2.3,3.9
NEWPDFILE 20.3,3.4.1,20.8
NEWSMFILE 19.3.1,3.4.1,19.4.6,19.5.3
NLIST 15.11.1,15.7,15.8,15.11.2,15.11.3
OBEY 17.4,4.2,17.1,17.9,A4.2.1
OFFER 3.6.1,3.2.5,3.4.2,3.4.4,3.5,3.6,3.6.2,3.7.1,19.5.4,20.2,A4.2.1
OPTION 17.9,14.5,16.6,17.1,17.2,17.4
OUTPUT 15.8.1,15.7.1,15.8,15.9.1
PACKHELP 6.5
parameter 2.4,2.5,8.5,16.2,16.3,17.2,17.3.4,17.4
checking 18.2,18.7,18.13
confidential 18.6.1
default 18.5.3
default value 2.5.4
description 18.3,18.5
file 17.2
integer 18.7,18.3
keyword 2.5.2,2.5,16.2,16.3,17.2,17.9,18.5.1,18.5.5,18.12
longreal 18.8,18.3
null 18.12
null value 18.5.3
obligatory 2.5.4,2.5.3,18.5.1,18.5.3
obtaining 18.3,18.2,18.6,18.8,18.10
omitting 2.5.1
optional 2.5,2.5.3,2.5.4
positional 2.5.1
previewing 18.10
prompt 2.5.3,16.2,17.2,17.4,18.3
remaining 18.10
string 18.3,18.6,18.6.1
value 18.5.2,18.6.1
PARM 7.2.2,7.1,7.5.7,8.2
PASCAL 10,4.5
PASSWORD 2.2.3,2.2.2
PDTOGROUP 20.9.2
PERMIT 3.7.1,3.2.5,3.5,3.7,3.7.2,3.7.4,20.2
prompts 2.5
QUIT 2.6
RECALL 5.9,5.1,14.7,17.9
RECAP 5.21,5.1,14.7,17.9
RECOVER 3.3
reissue 17.3.1,17.3
RELEASEMT 21.4.2
REMOVE 7.6.4,7.6.2,7.6.3,7.6.5
REMOVEDIR 7.5.6
RENAME 3.4.2,3.5,8.5,17.2,20.7,A4.2.1
repeating 17.3.3,17.3.4
RESETLOADER 19.5.4
RESTORE 3.9.3,3.9,3.9.4,3.9.6,14.8
RUN 7.4.1,7.1,7.4,7.5,7.5.2,7.5.7,7.6.3,8.2.1,8.3,9.3.3,9.5.6,18.1
SEARCHDIR 7.5.4,6.5,7.5.3,7.5.5,8.4.2
SEND 15.7.2,2.5.4,3.5,3.7.1,15.7,15.8,15.8.1,15.9
SETMODE 14.3.2
SHOW 5.20,5.1

STOP 2.6
structure 18.2
success 18.2
SUGGESTION 2.8
TAPEANAL 21.5.1
TELL 14.10
TERMINALTYPE 14.4,2.7.6,14.5,17.9
TIDYDIR 7.6.5
TOTAPE 21.5.3,21.5.2,21.6
TRANSFER 15.10,12.3,15.4,15.5
TRANSFERS 15.10.1,15.9.1
USEGROUP 3.2.4,3.3,3.4.3,3.4.4,3.5.1,3.6.2
USERS 2.4
VIEW 2.7.3,2.7.2,2.7.5,14.4,18.5.4
writing 18.1,7.4,9.3.3
in other languages 18.11
Command: prompt 2.3.3,2.3.6,2.4,17.3,17.3.2
communications network 1.5.5,2.2,2.3,14.2,14.2.1,14.2.3,15.3.2,A1.3.1,A1.3.3,A6
address 17.9
compiler 7.1,4.5,7.3.1,7.6.3,9.2,12.2,A1.3.2
listing 7.2.1,10
listing file 7.2.1,15.7.2
message 7.2.1
options 7.2.2,9.2
tables 8.2
compile-time 8.2
Computing Newsletter 2.7.1,6.5
Conc: prompt 4.5
CONCAT 4.5
concatenating files 4.5,7.2.1,15.7.2,18.6.1
input 13.6.2
concatenation operator 4.4,4.5,15.7.1,18.6
CONLIB 6.6
connect mode 19.4.1
READ 19.4.1,19.5.7
WRITE 19.5.6
connecting a file A4.2.1
console A6
constants 7.3.1
context editor 5.1
control character 8.6.2,17.9
control file 4.5,7.3.2
CONTROLCHARINTS (OPTION keyword) 17.9
CONVERT 4.7,4.5
COPY 4.4,2.5.1,3.2.4,3.4.1,3.7.1,4.5,20.5,20.8,A4.2.3
COPYTAPE 21.5.4
cpu A6
time 14.6.3
limit 7.4.1,17.7,A5
used 17.8
CPULIMIT 17.7,7.4.1,17.1
= 17.7
creating a file 3.4.1,3.7.1,3.8.1,4.4,18.7.1,A4.2.1
character 4.7,5.2,5.3,5.6.4,5.11,5.18,8.7.2,16.3
direct access 8.7.4
partitioned 20.3,20.8
member 20.5
store mapped 19.3.1
CTRL+A 17.9
CTRL+C 17.9

CTRL+K 17.9
CTRL+P 14.3.1,14.6.1
CTRL+R 2.3.5
CTRL+T 17.9
CTRL+X 2.3.5
CTRL+Y 2.3.5,2.5.3,17.2

data 8.2.2,17.4,18.1
  export 21.7,21.1
  import 21.7,15.4,15.5,21.1
  in-line 17.4
  input 15.1,15.3
  listing 15.1
data area 4.3
data entry 7.3.3,7.6.3,7.6.4,9.2,19.5.1
data file 13.4,3.3,4.2,4.4,4.5,8.7.1-8.7.3,9.5.1,9.5.3,13.2,13.6.4,18.4,18.6.1,18.7.1,20.2,21.5.3
  as output 9.5.4
  conversion 4.7
  fixed format 19.3,19.4.5
  format A6
  listing 15.7.1
  records 13.4
  unstructured 13.4.3,19.3,19.3.1,19.4.5,19.4.6
  variable format 19.3
Data: prompt 8.6.2
data reference 7.3.3
DATASPACE 19.5.2,19.5.1,19.5.3-19.5.5,19.5.7
date 3.9.3,3.9.6,4.2,17.8,18.6
DECKS (DETACH parameter) 16.3
DEFAULTDEFINESIZE (OPTION keyword) 17.9
DEFAULTTERMINALTYPE (OPTION keyword) 17.9
DEFINE 13.6,7.4.1,8.5,8.7.1,8.7.2,8.7.4,9.5.2,9.5.4-9.5.7,13.8-13.10,15.6,17.9,18.7.1, 19.3,19.4.1,19.4.5,19.4.6,20.5,20.6
  = 13.6.5
  C 9.5.4,13.6.4,18.8
DEFINEMT 21.4,13.7,21.4.1,21.5.3,21.6
DEL 2.3.5
DELETEDOC 15.9.1,15.7.1,15.9,15.11.3,16.5
deleting a character 2.3.5
deleting a file 3.8.1
deleting a line 2.3.5
DELIVER 15.7.3,16.3
  = 15.7.3
  text 8.6.4,A5
DELIVERY (DETACH parameter) 16.3
DESTROY 3.4.4,3.5,3.7.1,20.7,A4.2.1
DESTROYGROUP 3.4.5
DESTROYSET 3.4.6
DETACH 16.3,2.5.2,4.2,4.5,9.5.6,15.6,16.1,16.4,16.5,17.4
  AFTER 16.3
  DECKS 16.3
  DELIVERY 16.3
  LOG 9.5.6,16.3
  MAXLOG 16.3
  NAME 16.3
  ORDER 16.3
  PRIORITY 16.3
  RERUN 16.3
  TAPES 16.3
  TIME 16.3

UNIQUE 16.3
device driver A1.3.1
device queue 15.7.3,15.11.1
interrogating 15.8
device, remote 15.10.2
diagnostic tables 7.3.1
diagnostics 8.2,17.9
DIRECT A1.3.3,A6
direct access 9.5.1,13.4.1,13.6.3,20.6
direct access binary file 8.7
input/output 8.7.4
direct access file 3.7.1,9.5.5,19.2
Director A1.3.2,3.2,3.7.2,4.1,4.2,13.2,20.2,A5,A6
directory 7.5.4,7.5.6,7.6.3-7.6.5,8.2.1
active 7.6.2,7.5.3,7.6.3-7.6.5,17.6,A5
archive 8.2.1
list 7.5.4,7.5.5
session 7.5.3
directory file 3.3,4.2,4.3,7.5.2,7.6,7.6.5,8.4.1,18.6.1,21.5.3
creating 7.6.1,7.6.2
size 7.6.1
tidying 7.6.5
disc file A6
store A1.3.2
disc storage unit A6
disc store A1.3.2,A6
DISCARD 3.9.5
DISCONNECT 3.5.1,3.5
.ALL 3.5.1
displaying a line 2.3.5
document 15.6
queued 15.6.1,15.8
deleting 15.9,15.9.1,15.11.3
document printer 8.7.2,15.7
documentation
about VIEW 2.7.6
on-line 2.7.2
printed 2.7.1
DOCUMENTS 15.6.1,15.8,15.8.1
dot 21.4
double quotes 15.7.3
.DP 8.7.2
DPLIST 15.7
dynamic loading 7.7,8.2,9.2,9.3.3
DYNAMIC (PARM option) 7.7,8.2,9.2
dynamic reference 7.3.3,7.5.7,7.7.1

ECCE 5.11,5.1,8.9.2,13.3,14.10,16.2,17.4,20.5,20.8
%^ (correct mistake) 5.16
: prompt 5.12.2,5.12.3
> prompt 5.12.3,5.14.1,5.17,5.19.4
>> prompt 5.17
* 5.12.2
A (abstract) 5.17,5.18,5.20
A (adjust) 5.19.1
%A (format specification) 5.19.1
alternative command sequence 5.13
%B (backup) 5.16
B (break) 5.12.9
bracketing commands 5.13

C (case inversion) 5.12.8
C- (case inversion backwards) 5.12.8
%C (close) 5.12,5.12.2,5.16,5.17,5.20
calling EMAS commands 5.19.4
character manipulation 5.12.8
command failure 5.12.7
context specification 5.12.14
%D (control printing) 5.12.13
D (delete) 5.12.5,5.12.14
delimiting text 5.13
E (erase) 5.12.8
E- (erase backwards) 5.12.8
F (find) 5.12.2,5.12.5,5.12.14
%F (full monitoring) 5.12.11
file pointer 5.12.1
formatting text 5.19.1
G (get) 5.12.2
%H (help) 5.12.4
H (horizontal move) 5.12.8
handling non-printable characters 5.12.13
%I (extended macro) 5.14.1,5.19.4
I (insert) 5.12.5
indefinite repetition (* or 0) 5.12.2,5.13
inverted success/failure condition 5.13
J (join) 5.12.9
K (kill) 5.12.2
L (left shift) 5.12.8
%L (lower case terminal) 5.12.12
M (move) 5.12.2
M- (move backwards) 5.12.2
%M (normal monitoring) 5.12.11
macros 5.14
main file size 5.17
N (note) 5.17,5.18,5.20
O (orthography) 5.19.2
%O (secondary output) 5.18,5.20
optional execution 5.13
options 5.15
output file 5.12.2
P (print) 5.12.2
%P (profile) 5.15,5.12.13,5.19.1
private dictionary 5.19.2
programmed commands 5.13
Q (query spelling) 5.19.2
%Q (quit) 5.12,5.16
%R (retrieve) 5.16
R (right shift) 5.12.8
recovering from errors 5.16
repetition command 5.12.10
%S (secondary input) 5.17,5.20
S (substitute) 5.12.5
secondary input 5.17
secondary output 5.18,5.17
semicolon in command 5.12
simple subset of commands 5.12.2
spaces in command 5.12
spelling checker 5.19.2
start commands 5.14.1
syntax error 5.12
system dictionary 5.19.2

%T (terse mode) 15.12.11
T (traverse) 5.12.5,5.12.14
text location and manipulation 5.12.5
U (uncover) 5.12.5,5.12.14
%U (upper case terminal) 5.12.12
%V (IMP syntax checker) 5.19.3
V (verify) 5.12.5
%W (macro) 5.14
%X (macro) 5.14
%Y (macro) 5.14
%Z (macro) 5.14
ECCE#BACKUP 5.16
ECCE#START 5.14.1
:ECCE#TDICT 5.19.2
ECHO (OPTION keyword) 17.9
EDIT 5.2,2.5.1,3.4.1,3.5.1,3.7.1,5.1,5.3,13.3,14.10,16.2,17.3.3,17.4,20.8,A4.2.1
command 5.5
' (quote) 5.5
* 5.5,5.6.1
A (after) 5.5.1
B (bottom) 5.5.1,5.6.2
bracketed 5.6.1
C (cancel) 5.5.5
D (delete) 5.5.3
E (end) 5.5.6
F (file) 5.6.4
failure 5.5,5.6.1
G (go to) 5.5.1
H (hold) 5.5.1
I (insert) 5.5.2
J (justify) 5.6.5
K (kill) 5.6.2,5.6.3
L (layout) 5.6.5
M (move) 5.5.1
N (ignore word) 5.6.6
O (over) 5.6.2
P (print) 5.5.4
Q (quit) 5.5.6,5.5.7
R (remove) 5.5.3
repetition 5.5.5,5.6.1
S (separate) 5.6.2-5.6.4
sequence 5.5
spaces in 5.5
T (top) 5.5.1,5.6.2
U (uproot) 5.6.3
W (write) 5.5.7
X (mis-spelt word) 5.6.6
Y (correctly spelt word) 5.6.6
Z 5.5
cursor 5.4
moving 5.5.1,5.7
positioning 5.6.2
representation 5.5.4
extracting part of a file 5.6.4
<file+> 5.6.4
<filename> 5.5,5.5.2
formatting text 5.6.5
letter case 5.5
moving text within file 5.6.3
operation 5.7

output file 5.5.7,5.7
private dictionary 5.6.6
spelling checker 5.6.6
system dictionary 5.6.6
text string 5.5-5.5.4,5.6.3
Edit: prompt 5.4,5.8,5.9
editing a partitioned file member 20.5
editor 3.5,3.5.1,5.1,5.11,14.10,16.2,17.4,20.5,A1.3.2
screen 14.3.2,14.4
EDNET 2.2,2.3,2.3.1,14.2,14.2.3,15.3.2,15.7.1,A6
:E#EDITDICT 5.6.6
:E#EHA 5.5.7
elapsed time 8.8
electronic mail 15.12,15.1,A1.3.3
message 15.11.2
EMAS 1.1,15.1,17.9,A1.1
operating system 8.1
EMAS-3 2.2,2.3.4,2.7.1,8.1,8.6.1,15.3.1,A1.2
command language 17.1
commands 2.4,2.9
file system 3.2,13.2
hardware 6.7
EMAS 2900 2.7.1,8.1
EMAS3 (routine) 8.5,7.5,9.4.5,13.8,14.3.2
EMAS-A 2.2
EMAS3CLAIMCHANNEL 13.10,13.6.1
EMAS3CONNECT 18.9
EMAS3EXIST 8.6.5,9.4.6
EMAS3GETRESTOFLINE 18.10
EMAS3HELPINFO 18.5.4
EMAS3INTEGER 18.4,18.2,18.7,18.7.1,18.10,18.13
EMAS3INTERRUPT 8.6.3,9.4.4,14.6.4
EMAS3LASTPARAM 18.10
EMAS3LCLINE 18.10
EMAS3LINE 18.10
EMAS3LONGREAL 18.8,18.13
EMAS3PROMPT 8.6.2,8.6.1,9.4.1,9.4.3
EMAS3RENAME 18.9
EMAS3SETLINE 18.10
EMAS3SETRETURNCODE 18.9
EMAS3STRING 18.4,18.3,18.10,18.13
EMAS3UINFI 8.6.4,9.4.7,A5
EMAS3UINFS 8.6.4,9.4.8,A5
.END 7.3.2,18.6.1
end of file 2.3.5
entry name 4.2,7.6.3,18.11
entry point 4.3,7.3.1,7.5.5,7.7,18.2
error message 18.2,18.3,A4
log-on A4.1
exclamation mark 17.3,17.3.1,17.3.3
external data 8.2.2
external entry 8.2.1
external reference 7.7

failure message 8.3.1,8.7.2,18.9
FEP 14.2,A6
file 3.2.4,3.3,3.7,3.9.3,8.7.1,15.1,15.6,15.7.1,A6
access method 8.7.1

access permission 3.7,3.4.4,3.6.1,3.7.1,4.3,4.4,4.7,7.4.1,15.7.1,18.5.2,19.4.1,19.5.2,19.5.6, A6
default 3.7.1
everyone else's 3.7.1
full 3.7.1
interrogating 3.7.1
multiple 3.7.1
own 3.7.1
protect 3.7.1
read 3.7.1,18.4,18.6.1,18.7.1,A4.2.1
restricting 3.7.4
revoking 3.7.1
storage 3.7.3
to whom? 3.7.4
write 3.7.1,18.4,18.6.1,18.7.1
analysis 4.3
archived 3.9.2,3.9.3,21.1
access permission 3.9.6,3.9.3
destroying 3.9.5
backup 21.1
binary 12.3
character 13.3,2.5.1,3.3,3.4.1,3.5,4.2,4.4,4.5,5.3,5.6.4,5.11,7.2.1,8.7.1,8.7.2,9.5.1,9.5.3,13.2, 16.2,16.3,17.2,17.4,18.4,18.5.4,18.6.1,18.7.1,19.3,19.4.5,20.2,21.5.3
as output 9.5.4
creation 4.7,5.2,5.3,5.6.4,5.11,5.18,8.7.2,16.3
listing 15.7.1
overwriting 2.7.6,3.3,4.7,5.3,5.6.4,5.11,5.18,8.7.2,16.3
cherished 21.1
command 17.4,17.5
concatenation 4.5,7.2.1,15.7.2,18.6.1
connection 3.5,3.4.2,3.4.3,3.5.1,3.6.1,7.5.1,19.4.1,A2.3.2,A4.2.1,A6
content 3.4.3,4.1,4.2
control 4.5,7.3.2
copying 2.5.1,4.4,21.5.3,21.5.4
creation 3.4.1,3.7.1,3.8.1,4.4,18.7.1,A4.2.1
data 13.4,3.3,4.2,4.4,4.5,8.7.1-8.7.3,9.5.1,9.5.3,13.2,13.6.4,18.4,18.6.1,18.7.1,20.2,21.5.3
as output 9.5.4
conversion 4.7
fixed format 19.3,19.4.5
format A6
listing 15.7.1
records 13.4
unstructured 13.4.3,19.3,19.3.1,19.4.5,19.4.6
variable format 19.3
deletion 3.8.1
destroying 3.4.1,3.4.4-3.4.6,15.7,15.7.2
direct access 3.7.1,9.5.5,19.2
creation 8.7.4
direct access binary 8.7
input/output 8.7.4
directory 3.3,4.2,4.3,7.5.2,7.6,7.6.5,8.4.1,18.6.1,21.5.3
creating 7.6.1,7.6.2
tidying 7.6.5
disconnection 3.5,3.5.1
editing 5.2,5.11
existence 9.4.6,A4.2.1
existing 8.6.5,18.4,18.5.2,18.6.1,18.7.1
extending 5.6.4
format 19.2,19.7
full name A6

header 4.2,13.2,A6
index 3.2,3.2.3,3.2.4,3.9.1,3.9.3,A4.2.1,A6
permission 3.3
4.3
inserted 19.4.1
length 7.3.2
linked 7.2.1,13.3,15.7,15.7.2,15.8.1,15.9.1,15.11.2
listing 17.9
loaded 13.6.3
maximum size 3.2.2,3.2.1,3.3,3.4.3,3.6.2,3.7.1,3.9,3.9.1,3.9.5,4.1,8.6.5,15.2,
name 18.4,18.5.2,18.6,18.7,18.7.1,A6
non-standard 3.3,18.6.1,18.7.1
object 7.3.1,3.3,3.4.1,3.5,4.2,4.3,7.2.1,7.3.2,7.4,7.4.1,7.5.1-7.5.3,7.6,7.6.3,7.6.4,7.7,
8.2.2,8.2.3,8.6.2,9.2,9.3.1,9.3.3,10,12.2,18.6.1,18.7.1,18.11,20.2,21.5.3,A6
accessing 8.2.1
altering 7.3.3
destroyed 7.6.4,7.6.5
history 4.3,7.3.1
linked 4.3,4.6
linking 7.3.2
overwriting 7.2.1,7.3.3
structure 4.3
on-line 3.2,A6
output 17.9,A1.3.3
overwriting 3.4.1,3.7.1,7.2.1,13.10,18.7.1,A4.2.1
partitioned 20.1,3.2.5,3.3,4.2,5.3,5.11,7.2.1,15.7.2,20.9.1,20.9.2,21.5.3
access permission 3.7.1
header 20.2
member 4.3,4.4,7.2.1,7.4.1,18.4,18.6.1,19.4.3,19.5.2,19.5.7,20.2
member naming 20.4
member type 20.8
structure 20.2
protection 3.4.1,3.4.4,3.5.1,3.7.1,3.8.1
record format
F 18.7.1
V 18.7.1
renaming 3.4.2,3.4.1,3.9,3.9.3
sequential binary 8.7
input/output 8.7.3
size 3.7.1,8.7.4,9.5.5,13.2,18.7.1,19.3.1,A4.2.1,A5,A6
changing 19.4.6
source 7.2.1,10,13.3
status 3.8.1,3.3,3.4.2,3.4.3,3.9.3,4.4,4.7,20.9
store mapped 3.7.1,4.2,8.7.5,19.2-19.3.1,19.4.1,19.5.1,19.5.2,20.6,A2.3.2
creation 19.3.1
overwriting 19.4.1,19.4.2,19.5.1,19.5.3,19.5.6
structure 4.2,13.2
Subsystem work 17.9
system A1.3.3,A6
temporary 18.6.1
transfer 15.1,15.3.1,15.3.2,15.9.1,15.10,A1.3.1,A1.3.3,A6
information 15.10.1
transferring ownership 3.6
type 3.4.1,3.4.3,4.1-4.3,13.2,18.5.2,18.6.1
File: prompt 4.5,17.2
filename A4.2.1
FILES 3.3,2.4,3.4.6,3.9.2,13.3
filestore 1.5.3
fixed format record 13.4.1,13.6.4
foreground mode 8.7.2,8.8,16.1,A2.3.3,A6

foreground password 2.2.2,2.2.3
foreground process A4.1
foreground session 3.9.4,17.4,17.9
form feed 13.3
FORMAT (DEFINE parameter) 9.5.7
format effector 9.5.4
FORMAT (FORTRAN statement) 9.5.1,9.5.3,9.5.4
FORTRAN 7.2.1,2.5.1,3.9.4,4.5
BLOCK DATA 7.3.1,9.2,9.3.2,19.5.1
CALL statement 9.3.1,9.4.1,9.4.5-9.4.8,18.3
calling IMP 8.2.3,9.4.9
character*1 A6
CHARACTER variables 9.4.1,9.4.3-9.4.6
command 18.3
COMMON area 7.5.7
COMMON block 7.3.1,9.3.2,19.1,19.2,19.5.1-19.5.7
COMMON statement 9.2
COMPLEX variables 9.2
DATA statement 7.3.1
DOUBLE PRECISION variables 9.2
ENTRY entry 9.2
EQUIVALENCE statement 9.2
format effector 15.7.1
FORMAT statement 9.5.1,9.5.3,9.5.4
formatted I/O 21.4.3
FUNCTION entry 9.2
graphics 9.3.5
INTEGER variables 9.2
LOGICAL variables 9.2
MAIN PROGRAM entry 9.2
NAG library 9.3.5
OPEN statement 9.5.5,9.5.7
parameter passing 8.6.1
PARM options 9.2
program 7.7,9.1,9.3.3,9.5.2,9.5.6
READ 18.4,18.7.1
REAL variables 9.2
STOP 9.3.3
subroutine 7.3.1
SUBROUTINE entry 9.2
system library 9.3.4
unformatted access 21.4.3
WRITE 18.7.1
Fortran 7.1,12.3,13.8,13.10,18.1,18.2,18.11
FROMTAPE 21.5.2,21.5.3,21.6
Front End Processor 14.2,17.9,A6
FSTARTFILE (OPTION keyword) 17.9
FTRANS A1.3.3,A6
full stop 3.2.2,3.2.3,3.3.3,3.9,15.6.2
FUNDS A5

general linkage area 19.5.1
GLA 19.5.1
Global Controller A1.3.1,A2.3
GP 15.6.2
GPLIST 15.7
graph plotter 15.1,15.6.2,15.7

group 3.2,3.2.1,3.2.4,3.3,3.4.3,3.4.4,3.5.1,3.6.1-3.7,3.8.1,3.9,4.3,4.4.1,20.9-20.9.2,A5,A6
access permission 3.7.2,3.2,3.2.3,A6
copying 4.4.1
creating 3.2.3,3.9
destroying 3.4.5,3.4.6
existing 18.6.1
name 3.2.1-3.2.4,3.3,18.6,A6
renaming 3.4.2
status 3.2,3.2.3
GROUPTOPD 20.9.1

hard copy terminal 2.3,2.3.5,2.7.6
hardware A6
failure 3.8.1
HAZARD 3.8.1
head crash A6
HELP 2.7.4,2.7.2,2.7.6,6.5,14.4
DEVICES 15.6.2,15.7.1,15.11.1
EDIT 5.1
FORTRAN 2.7.4
GRAPHICS 6.5
hexadecimal A6
hierarchical filestore 20.1,20.9
HOLDMT 21.4.2
host 2.2,15.10.2,15.11.1,15.11.2,15.11.3,A1.3.3
HOSTS 15.10.2

I8 (PARM option) 9.2
IBM OS/MVS 6.7
IMP 7.2.1,3.4.1,4.5,7.1,8.1,13.8,13.10,17.3.1,18.1,18.2,20.6,A2.3.2,A4.2.1,A6
accessing Subsystem procedures 8.6
arrays 8.8
begin/end 5.19.3
byteinteger A6
calling FORTRAN 8.2.3,9.4.9
CHANGESM 19.4.6
CLOSEDA 8.7.4
CLOSESM 19.4.4,19.4.6
CLOSESQ 8.7.3
constant array 8.8
constantinteger 7.3.1
cycle/repeat 5.19.3
dynamicroutinespec 7.7,8.2
efficiency on EMAS 8.8
executing programs 8.3
external spec 8.6.1
external specification 8.3.1
externalroutinespec 7.7
formatting source 8.9.1
input/output 8.7
monitor 8.3.1
OPENDA 8.7.4
OPENSM 19.4.1,19.4.5,19.4.6
OPENSQ 8.7.3
own array 8.8
own variable 7.3.1
PARM options 8.2
program 7.3.1,7.7,8.6
READCH 8.7
READDA 8.7,8.7.4

READLSQ 8.7.3
READSQ 8.7,8.7.3
READSYMBOL 19.4.5
SELECT INPUT 18.4,18.7.1
SELECT OUTPUT 8.7.2,18.7.1
sequential access 21.4.3
spec 8.2.1,8.2.2
start/finish 5.19.3
stop 8.3.1
syntax checker 5.19.3,8.9.2
System library 8.4.1
unsatisfied references 8.3.1
WRITE 8.7
WRITEDA 8.7,8.7.4
WRITESQ 8.7,8.7.3
.IN 8.7.2,9.5.1,9.5.3,9.5.6,13.6.2
incarnation A5
index
archive 3.9.3,A6
file 3.2,3.2.3,3.2.4,3.9.1,3.9.3,A4.2.1,A6
permission 3.3
process A6
INFO A4.1
initialisation values 7.3.1,7.5.1
INITPARM 7.2.3,7.1
= 7.2.3
NOARRAY 7.2.3
NOCHECK 7.2.3
INITWORKSIZE (OPTION keyword) 17.9
input 13.6.2
'Input Ended' 13.6.2
INSERT 7.6.3,7.6.2,7.6.5,8.2.1,17.2
Int: prompt 8.6.3,14.6.1,17.9
interactive terminal 2.3,2.2.2,2.2.3,2.3.1,2.3.4,2.6,2.7.2,4.3,4.5,8.6.2,8.6.3,8.7.2,9.5.4,13.6.2, 14.1,14.2.3,15.3,15.3.1,15.7-15.7.2,15.8.1,16.1-16.4,17.4,18.1,18.2,A1.3.2,A6
address 14.5,17.9
blank lines 17.9
connect time 17.8
input buffer 15.3.1,17.9
interrupts 14.6
I/O buffers 14.5
I/O operation 5.9,5.21
output buffer 17.9
transactions 14.7,17.9
type 14.4,14.5
interrupt 9.4.4,12.3,A6
generating 14.6.1,14.6.2
meaning 14.6.3
multi-character 8.6.3,14.6.1,14.6.4
single character 8.6.3
I/O A6
device A6
IPL A6
ITINSIZE (OPTION keyword) 17.9
ITOUTSIZE (OPTION keyword) 17.9
ITWIDTH A5

job
background 9.4.3,9.4.4,9.5.6
file 8.7.2,16.1-16.3
journal 8.7.2

KERMIT 15.3.2,15.3.1

L8 (PARM option) 9.2
laser printer 15.7
LAYOUT 5.2
LET (LOADPARM option) 7.5.7,9.3.3
library 6.2,A1.3.2
contributed 6.6
graphics 8.4.2
IMP System 8.4.1
import 6.3
line printer 2.7.2,8.7.2,13.6.2,15.1,15.6.2,15.7,15.7.2,16.3,A1.3.1,A1.3.3,A6
LINK 7.3.2,4.6,7.1
MAP 7.3.2
link map 7.3.2
Link: prompt 7.3.2
linked file 7.3.2
LIST 15.7.1,3.5,4.5,9.5.4,15.7,15.7.2,15.8,15.8.1,15.9,15.11.1,20.6,A4.2.1
FE 15.7.1
listing file 7.2.1
load data 7.3.1,7.3.3,9.3.1
load module 6.7
load time 7.3.1
loader 7.6.3
LOADPARM 7.5.7,7.5.2
= 7.5.7
LET 7.5.7,8.3.1,9.3.3
MIN 7.5.7,7.7
ZERO 7.3.1,7.5.7
Local Controller A1.3.1,A1.3.2
localname 3.2.1-3.2.4,A6
LOG (DETACH parameter) 9.5.6,16.3
logging-off 2.6
logging-on 2.3.3
logical channel 13.6.1,13.5,20.5,20.6,21.4
in program 13.10
number 8.7.1,18.4,18.7,18.7.1
0 8.7.2
log-off 3.5
log-on 17.9,A1.3.3
LOOK 5.8,5.1,5.3,20.8
Look: prompt 5.8
LOOKFOR 7.5.5
.LP 8.7.2,9.5.1,9.5.2,9.5.4,13.6.2,13.8,15.7.2,15.8.1,16.3,20.6,21.5.1
LP 15.6.2

magnetic tape 3.9,8.7.3,9.5.1,12.3,13.7,15.4,15.5,21.1,A1.3.1,A2.2.2,A6
analysis 21.5.1
blocksize 21.4,21.4.1,21.5.1-21.5.4
BT 21.4.2
character code 21.4.3,21.5.2
EMAS ISO to IBM EBCDIC 21.5.3
IBM EBCDIC to EMAS ISO 21.5.2
density 21.2
error messages 21.6

file copying 21.5.2
filename 21.4,21.5.4
format 21.4.1
hardware standard 21.3.1
identification 21.3.2
inter-block gap 21.5.1
label 21.4,21.4.1,21.5.1,21.5.3,21.5.4,21.6
labelling 21.2
mark 21.4.1,21.5.4
multiple files 21.5.2,21.5.3
name 21.5.1-21.5.4
private 16.1,16.3,21.2,21.5,21.5.3
record format 21.4,21.5.2,21.5.3,21.6
software standard 21.3.3,21.5.4,21.6
unlabelled 21.5.3,21.5.4,21.6
access 21.4.1
unloading 21.4.2
volume label 21.3.2
volume number 21.4
write ring 16.3,21.4
MAIL 15.12,2.3.6,5.6.5,14.8,14.10
bulletin board 15.12
Mail: prompt 2.3.6,15.12
MAILER 14.8,A1.3.3,A6
main program entry 18.11
main store A1.2,A1.3,A1.3.1
MAKECOMMAND 18.11
MANAGR A6
MAP (LINK keyword) 7.3.2
MAXDICT (PARM option) 8.2
maximum file size 13.6.3
MAXLOG (DETACH parameter) 16.3
memory A6
message 3.9.3,14.8-14.10,A5
creating 15.12
editing 15.12
forwarding 15.12
listing 15.12
receiving 15.12
replying 15.12
sending 15.12
MESSAGES 14.9,14.10
METER 17.8,2.4,2.7.6,7.5.5,17.1,17.3
microcomputer 2.3,15.3,15.3.1,17.9
MIN (LOADPARM option) 7.5.7,7.7
-MOD 2.7.6,5.18,21.4
modem 14.2.2,A6
MODIFY 7.3.3,7.1,7.7
multi-access system A2.2.2

NAME (DETACH parameter) 16.3
NDELETEDOC 15.11.3
NDOCUMENTS 15.11.2,15.11.3
network name A5
NEWDIRECTORY 7.6.1,3.4.1
NEWGEN 3.4.3,3.7.1
NEWGROUP 3.2.3,3.9
NEWPDFILE 20.3,3.4.1,20.8
NEWSMFILE 19.3.1,3.4.1,19.4.6,19.5.3
NLIST 15.11.1,15.7,15.11.2,15.11.3

NOARRAY (PARM option) 7.2.3,8.2,9.2
NOCHECK (PARM option) 7.2.3,7.3.1,8.2,9.2
NOCOMMENTS (PARM option) 9.2
NODIAG (PARM option) 8.2,9.2
NOLINE (PARM option) 8.2
NOLIST (PARM option) 8.2,9.2
non-printable characters 5.12.13,13.3
non-standard file 3.3
NOTRACE (PARM option) 8.2
NOWARNINGS (PARM option) 9.2
.NULL 5.3,5.8,7.2.1,9.5.5,13.6.2,16.3,18.6.1,19.4.1,21.5.2,21.5.3

OBEY 17.4,4.2,17.1,17.9,A4.2.1
object file 7.3.1,3.3,3.4.1,3.5,4.2,4.3,7.2.1,7.3.2,7.4,7.4.1,7.5.1-7.5.3,7.6,7.6.3,7.6.4,7.7, 8.2.2,8.2.3,8.6.2,9.2,9.3.1,9.3.3,10,12.2,18.6.1,18.7.1,18.11,20.2,21.5.3,A6
accessing 8.2.1
altering 7.3.3
destroyed 7.6.4,7.6.5
history 4.3,7.3.1
linked 4.3,4.6
linking 7.3.2
overwriting 7.3.3
structure 4.3
Object: prompt 7.3.2
OCP type A5
OFFER 3.6.1,3.2.5,3.4.2,3.4.4,3.5,3.6,3.6.2,3.7.1,19.5.4,20.2,A4.2.1
off-line A6
on-line A6
OPENSM (IMP routine) 19.4.1,19.4.5,19.4.6
operating system 1.1,A2.2.1,A6
Operations Manager 15.2
OPT (PARM option) 8.2,9.2
OPT1 (PARM option) 9.2
OPT2 (PARM option) 9.2
OPTEXT (PARM option) 9.2
OPTION 17.9,14.5,17.1,17.4
= 17.9
? 17.9
ARRAYDIAG 17.9
BLANKLINES 17.9,14.5
BSTARTFILE 17.9,16.6,17.4
CONTROLCHARINTS 17.9,14.5,14.6.2
DEFAULTDEFINESIZE 17.9
DEFAULTTERMINALTYPE 17.9,14.5
ECHO 17.9
FSTARTFILE 17.9,17.4
INITWORKSIZE 17.9
ITINSIZE 17.9,14.5,15.3.1
ITOUTSIZE 17.9,14.5
PARITIES 17.9
PRELOADFILE 17.9
RECALL 17.9,14.5
NONE 14.7
PERM 5.9,5.21,14.7,17.2
TEMP 5.9,5.21,14.7
USTACKSIZE 17.9
Options(): prompt 17.2
ORDER (DETACH parameter) 16.3
.OUT 3.3,7.2.1,8.7.2,9.5.1,9.5.6,13.6.2,15.7.1,15.7.2,17.4,18.4
Out() prompt 17.2

OUTPUT 15.8.1,15.7.1,15.8,15.9.1
output 13.6.2,13.6.4,15.7.3,20.9
  device 8.7.1,8.7.2,15.1,15.6,15.7,15.7.1,15.11.1,15.12
    name 15.6.2
    queue 15.6,15.8.1,15.11.2
    type 15.6.2
  queue 15.7.1
'Output exceeded' 16.3
overwriting a character file 2.7.6,3.3,4.7,5.3,5.6.4,5.11,5.18,8.7.2,16.3
overwriting a file 3.4.1,3.7.1,7.2.1,13.10,18.7.1,A4.2.1
overwriting a store mapped file 19.4.1,19.4.2,19.5.2,19.5.3,19.5.6
overwriting an object file 7.2.1,7.3.3

package 6.2,6.1,6.5,6.9
  import 6.3
packet A6
Packet Assembler/Dis-assembler 14.2.1,14.2,14.2.2
PACKHELP 6.5
PAD 14.2.1,2.3.1,2.3.2,2.3.4,8.7.2,14.2,14.2.2,14.2.3,A6
  controlling 14.3
  instructions to 14.3.2
PAD> prompt 2.3.2,14.3.1
page 4.2,A1.3.1,A1.3.2,A6
  fault A2.3.1,A2.3.3
  frame A6
  of store A2.3,A2.3.1
  turn 8.8,14.6.3,17.8,A6
paged System process A6
paging A2.3,A1.3.1,A2.1,A2.3.1,A2.3.2,A6
Paging Manager A1.3.1
PAM 18.3
  routine format 18.3
  vector 18.3,18.5
    default field 18.5.3,18.5.6
    help field 18.5.4,18.5.6,18.12
    keyword field 18.5.5
    missing field 18.5.6
    prompt field 18.5.1,18.5.5,18.5.6
    qualifier field 18.5.2,18.5.6,18.6,18.7,18.13
    qualifier tag 18.6.1,18.7.1
paper tape 15.2
Parameter Acquisition Mechanism 18.3,13.5
PARITIES (OPTION keyword) 17.9
parity bit 17.9
PARM 7.2.2,7.1,7.5.7,8.2,9.2
  = 7.2.2
  ATTR 9.2
  CODE 8.2,9.2
  DYNAMIC 7.7,8.2,9.2
  I8 9.2
  L8 9.2
  MAXDICT 8.2
  NOARRAY 7.2.3,8.2,9.2
  NOCHECK 7.2.3,7.3.1,8.2,9.2
  NOCOMMENTS 9.2
  NODIAG 8.2,9.2
  NOLINE 8.2
  NOLIST 8.2,9.2
  NOTRACE 8.2
  NOWARNINGS 9.2

OPT 8.2,9.2
OPT1 9.2
OPT2 9.2
OPTEXT 9.2
R8 9.2
STANDARD 9.2
XREF 9.2
partitioned file 20.1,3.2.5,3.3,4.2,5.3,5.11,7.2.1,15.7.2,20.9.1,20.9.2,21.5.3
access permission 3.7.1
creating 20.3,20.8
header 20.2
member 3.3,4.3,4.4,7.2.1,7.4.1,18.4,18.6.1,19.4.3,19.5.3,19.5.7,20.2
creating 20.5
editing 20.5
naming 20.4
type 20.8
structure 20.2
PASCAL 10,4.5
Pascal 10,12.3,A2.3.2
PASSWORD 2.2.3,2.2.2
password 2.2.2,2.2.1,2.3.2,2.3.3,A1.3.3,A4.1,A6
background 15.2
foreground 2.2.2,2.2.3
transfer 15.11.2,15.11.3
PDTOGROUP 20.9.2
PERMIT 3.7.1,3.2.5,3.5,3.7,3.7.2,3.7.4,20.2
PRELOADFILE (OPTION keyword) 17.9
printable characters 13.3
PRIORITY (DETACH parameter) 16.3
procedure name 7.3.1,7.5.1-7.5.3,7.6,7.6.3,7.6.4
process A1.3,A6
DIRECT A1.3.3
foreground A4.1
FTRANS A1.3.3
index A6
MAILER 14.8,A1.3.3
paged A1.3.2,A1.3,A1.3.1,A1.3.3
System A6
SPOOLR A1.3.3
start time A5
Supervisor A1.3.1,A1.3,A1.3.2,A6
System A1.3.2,A1.3.3
user A1.3.2,A6
virtual A6
VOLUMS 14.8,A1.3.3
processor A1.2
program 6.2,8.6.2,8.7.1,A4.2.3
environment 9.4.6,A5
execution 7.5,18.1
failures A1.3.2
FORTRAN 3.5
IMP 8.2.1,8.6
efficiency 8.8
executing 8.3
interface 9.4.2
large A2.3.3
linking 7.3.2
loader 7.3.3,7.5,7.5.2,7.5.7,19.5.1,A1.3.2
loading 7.7,A6
overlay A2.3.2

portability 9.4,12.3,19.6
storage A2.2.1,A2.3.1
user stack 17.9
programming languages 8.6.1,12.1,18.1
prompt 8.6.2,12.3
:, ECCE 5.12.2,5.12.3
>>, ECCE 5.17
>, ECCE 5.12.3,5.14.1,5.17,5.19.4
Command: 2.3.3,2.3.6,2.4,17.3,17.3.2
command 2.5
parameter 2.5.3,16.2,17.2,17.4,18.3
Conc: 4.5
Data: 8.6.2
Edit: 5.4,5.8,5.9
field, PAM vector 18.5.1,18.5.5,18.5.6
File: 4.5,17.2
Int: 8.6.3,14.6.1,17.9
length 8.6.2
Link: 7.3.2
Look: 5.8
Mail: 2.3.6,15.12
Object: 7.3.2
Options() 17.2
Out() 17.2
PAD> 2.3.2,14.3.1
Quit: 5.5.6
Recall: 5.9
Subsystem 16.2
text 9.4.3,A5
View: 2.7.6,5.12.4

question mark 2.5.3,2.5.4
QUIT 2.6
Quit: prompt 5.5.6

R8 (PARM option) 9.2
READSYMBOL (IMP routine) 19.4.5
RECALL 5.9,5.1,14.7,17.9
RECALL (OPTION keyword) 17.9
Recall: prompt 5.9
RECAP 5.21,5.1,5.14.1,14.7,17.9
RECAP#START 5.14.1
record length 9.5.5
RECOVER 3.3
reference name 9.3.1
RELEASEMT 21.4.2
REMOVE 7.6.4,7.6.2,7.6.3,7.6.5
REMOVEDIR 7.5.6
CONLIB:GENERAL 7.5.6
ERCLIB:GRAPHICS 7.5.6
ERCLIB:NAGLIB 7.5.6
RENAME 3.4.2,3.5,8.5,17.2,20.7,A4.2.1
RERUN (DETACH parameter) 16.3
RESETLOADER 19.5.4
RESTORE 3.9.3,3.9,3.9.4,3.9.6,14.8
return code 18.2
setting 18.9
route A6
route (PAM vector qualifier) 18.7,18.7.1
RUN 7.4.1,7.1,7.4,7.5,7.5.2,7.5.7,7.6.3,8.2.1,8.3,9.3.3,9.5.6,18.1

run-time 8.2
diagnostics 9.2

Scheduler A1.3.1
scheduling A1.3.1
SCRIBE 5.2
SEARCHDIR 7.5.4,7.5.3,7.5.5,8.4.2
= 7.5.3,7.5.4
CONLIB:GENERAL 7.5.4
ERCLIB:NAGLIB 7.5.4
list 7.5.4,8.2.1
MICROS:EMASLIB 15.3.1
PLULIB:PACKDIR 6.5
searchlist 7.5.2,7.5.3,7.5.6
segment A6
semicolon 18.3,18.5.5
SEND 15.7.2,2.5.4,3.5,3.7.1,15.7,15.8,15.8.1,15.9
sequential access 9.5.1,13.4.1,13.6.4
sequential binary file 8.7
input/output 8.7.3
sequential input 9.5.3
sequential output 9.5.4
service name A5
Service Support Unit 2.2.1
session directory 7.5.3
SETMODE 14.3.2
= 14.3.2
?? 14.3.2
CONTROLCHARINTS 14.3.2,14.6.2
ECHO 14.3.2
GRAPH 14.3.2
MESSAGE 14.3.2
NATIVE 14.3.2
NOECHO 14.3.2
TRANSPARENT 14.3.2
shared code 7.3.1
SHOW 5.20,5.1,5.14.1
SHOW#START 5.14.1
SOAP 8.1,8.9.1
software A6
commercial 6.8
failure 3.8.1
IMP 8.2.1
import 6.4
selection 6.9
source file 7.2.1,10
spelling checker 5.6.6,5.19.2
SPOOLR 15.7.1,15.7.2,A1.3.3,A6
SS# 3.3
:SS#DIR 7.6.2,17.6
stack 8.2,A2.3.2,A6
user program 17.9
STANDARD (PARM option) 9.2
static reference 7.7.1
STOP 2.6
store A6
archive 3.9,3.2,3.7,3.9.1,3.9.3,3.9.5,14.8,A1.3.3,A6
files 3.9.2
backing A2.3-A2.3.2
backup 3.9,14.8,A1.3.3,A6

conventional A2.3.3
fragmented A2.2.1
main A2.2.1-A2.3.2,A6
paged A2.3.3
store mapped file 3.7.1,4.2,8.7.5,19.2-19.3.1,19.4.1,19.5.1,19.5.2,20.6,A2.3.2
store mapping 8.7,12.3,13.4.3,13.6.3
SUBSYS:ERRORTEXTS A4.2
Subsystem 1.4,3.2,4.1,4.2,7.5,8.1,8.5,8.6.3,13.2,13.4.3,13.6.1,13.6.3,14.4-14.6,15.6,17.1,17.3.2,17.9,19.3,20.2,A1.3.2,A4.2.3,A6
basefile 7.5.3
command 20.6,21.4
level 5.5.6
components 7.5.3
Director A6
environment 9.4.6,A5
error messages A4.2
failures A1.3.2
interface 8.6.1,9.1,12.2
loader 12.2
obtaining information 8.6.4
procedure 7.5.3,9.4.2
failure 18.9
prompt 16.2
work file 17.9
SUGGESTION 2.8
Supervisor A1.3.1,A1.3,A1.3.2,A2.3.1,A5,A6
System 3.5.1,3.9.1,8.6.4,A1.2,A1.3,A1.3.2,A1.3.3,A6
System Manager 3.2.2,3.8.1,A4.1,A6

T# 3.2.4,3.3
TAPEANAL 21.5.1
TAPES (DETACH parameter) 16.3
TCP 2.3.1,2.3.4,14.2,A6
TELL 14.10,5.19.4
.TEMP 13.6.2
temporary file 13.6.2,18.6.1
terminal A6
address 14.5,A5
concentrator 14.2.3
hard copy 2.3,2.3.5,2.7.6
interactive 2.3,2.2.2,2.2.3,2.3.1,2.3.4,2.7.2,9.5.4,13.6.2,14.2,14.2.3,16.1-16.4,A6
interrupts 14.6
I/O buffers 14.5
I/O operation 5.9,5.21
transactions 14.7
type 2.7.6,14.4,14.5
Terminal Control Processor 14.2
TERMINALTYPE 14.4,2.7.6,17.9,A5
text formatting 5.2,5.6.5,5.19.1
TIDYDIR 7.6.5
time 4.2,17.8
TIME (DETACH parameter) 16.3
'Time Exceeded' 16.3
T#LIST 2.5.4,5.8,5.20,7.2.1,15.7.2
T#OBJECT 7.2.1,7.4.1
TOTAPE 21.5.3,21.5.2,21.6
track A6
TRANSFER 15.10,12.3,15.4,15.5
TRANSFERS 15.10.1,15.9.1
typing ahead 2.3.6

UNIQUE (DETACH parameter) 16.3
UNIX 3.2.3
unshared code 7.3.1
USEGROUP 3.2.4,3.2.2,3.3,3.4.3,3.4.4,3.5.1,3.6.2,5.14.1
  = 3.2.4
user 3.3,3.6.2,3.9.6,A1.2
User Notes 2.7.1,6.5,15.3.1,15.3.2,15.10.1,Bib
user process 8.6.4,A1.3.2,A2.3,A6
user stack size A5
usergroup 3.7.1,3.7.2,3.7.4,3.9.6,A6
username 2.2.2,2.2.1,2.3.2,2.3.3,3.2–3.2.2,3.2.4,3.6.1,3.7.1,8.6.4,15.2,15.11.1,15.11.2,15.11.3, 18.6,A4.1,A5,A6
USERS 2.4
USTACKSIZE (OPTION keyword) 17.9

variable format record 13.4.2,13.6.4
VDU 2.3,A6
VIEW 2.7.3,2.7.2,2.7.6,5.12.4,14.4,18.5.4
View: prompt 2.7.6,5.12.4
virtual address A1.3.2,A2.3.2,A6
virtual machine A1.3.2
virtual memory 1.5.4,3.4.2–3.4.4,3.5.1,3.6.1,7.3.1,7.5,7.5.1,19.2,19.4.1,19.4.4,19.4.6, A1.2,A1.3.1,A1.3.2,A2.1,A2.3,A2.3.2,A4.2.1,A6
virtual processor A1.3.1
Virtual Storage Simulator 6.7
visual display unit 2.3,2.7.6
VOLUMS 3.9.3,14.8,A1.3.3,A6
VSS 6.7

warning messages 7.5.2
wild-card 17.2,7.6.3,7.6.4,17.1,18.6.1
  parameter 5.14.1
word processing package 15.3

XREF (PARM option) 9.2
XTALK 15.3.1,15.3.2

ZERO (LOADPARM option) 7.3.1,7.5.7