L1 MOS

L1 MOS

DMS - DATA MANAGEMENT SYSTEM

User Guide

olivetti

PREFACE

This manual describes the operating standards valid in the DMS environment and the programming by using DML (Data Manipulation Language).

This manual is principally intended for the application user and also for the person responsible for the installation of the DMS, i.e. the data administrator.

SUMMARY

This manual describes the following subjects:

- DMS operating environment
- DMS language elements
- The command file
- The procedures
- Error handling
- DML commands

This manual also includes:

- An example of DML programming
- Interactive specification of commands

The appendix contains a list and description of the error messages.

REFERENCES:

Read first...

Introduction to MOS
Code 4002130 G (vol. 2)

DMS - Features and Functions
Code 4002610 Q (vol. 2)

For further information, read...

DMS - Data Definition Language
User Guide
Code 4004570 D

MOS - EDITOR Reference Manual
Code 4002440 P (vol. 6)

MOS SHELL - Commands Reference Manual
Code 4002770 Q (vol. 3)

Glossary/Glossario
Code 4002140 H (vol. 1)

First Edition: January, 1983
Release 1.0

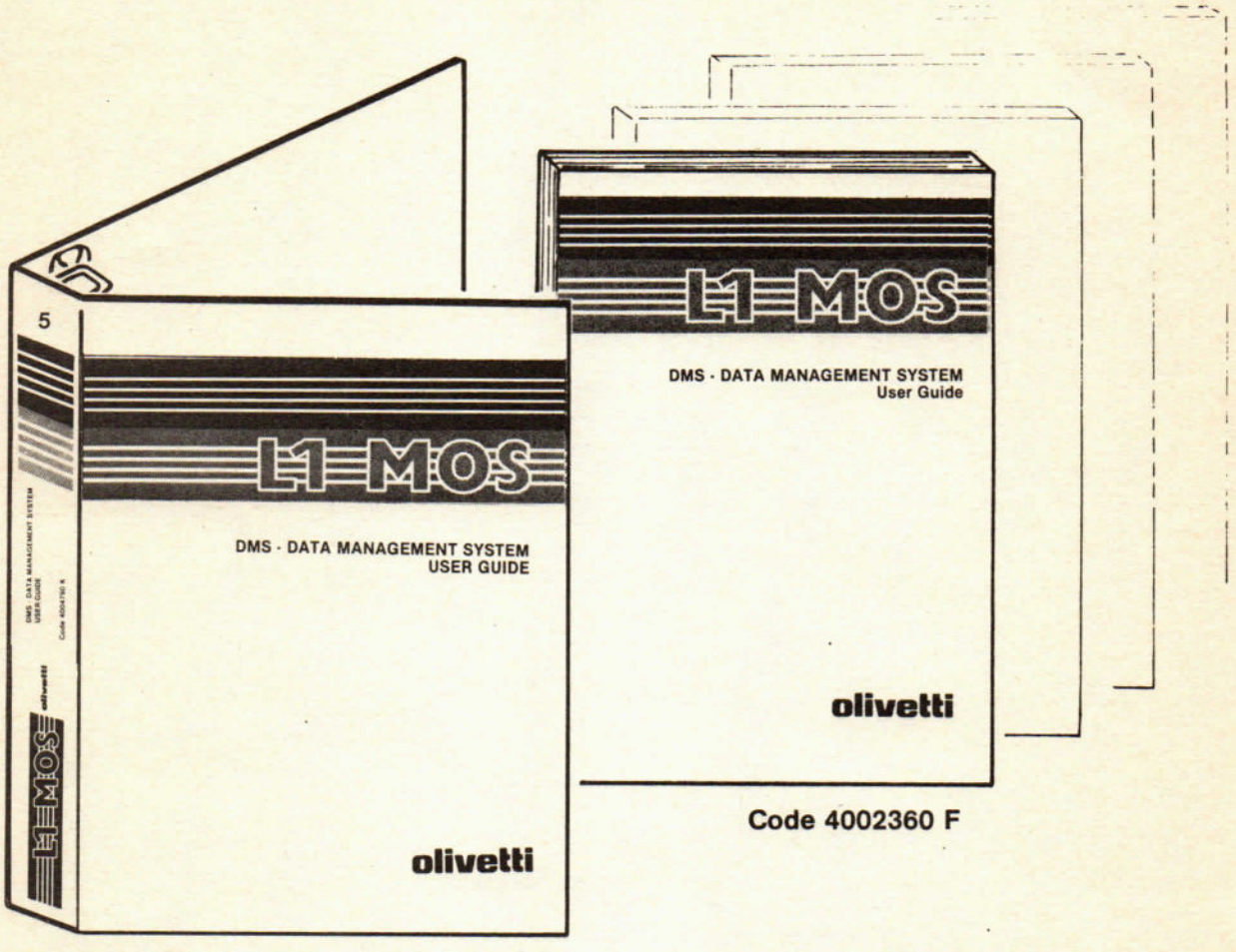
First Update: December, 1983
Release 2.0

Second Update: March,1984
Release 3.0

PUBLICATION ISSUED BY:

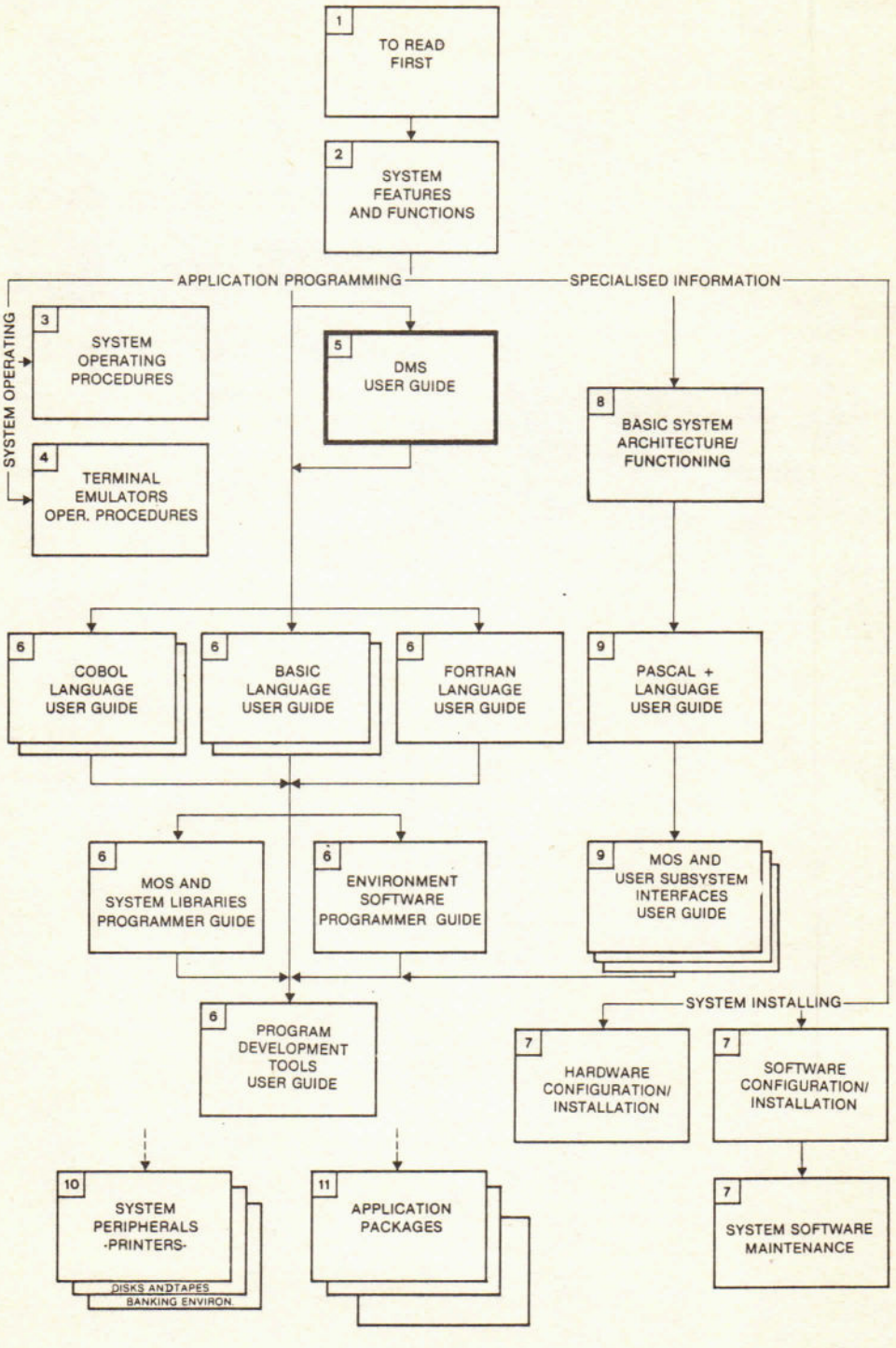
Ing. C. Olivetti & C., S.p.A.
Direzione Documentazione
77, Via Jervis - 10015 IVREA (Italy)

4002362 L



Code 4004790 K

Code 4002360 F


1
TO READ FIRST
2
SYSTEM FEATURES AND FUNCTIONS
APPLICATION PROGRAMMING
SPECIALISED INFORMATION
SYSTEM OPERATING
3
SYSTEM OPERATING PROCEDURES
4
TERMINAL EMULATORS OPER. PROCEDURES
5
DMS USER GUIDE
8
BASIC SYSTEM ARCHITECTURE/ FUNCTIONING
6
COBOL LANGUAGE USER GUIDE
6
BASIC LANGUAGE USER GUIDE
6
FORTRAN LANGUAGE USER GUIDE
9
PASCAL + LANGUAGE USER GUIDE
6
MOS AND SYSTEM LIBRARIES PROGRAMMER GUIDE
6
ENVIRONMENT SOFTWARE PROGRAMMER GUIDE
9
MOS AND USER SUBSYSTEM INTERFACES USER GUIDE
6
PROGRAM DEVELOPMENT TOOLS USER GUIDE
SYSTEM INSTALLING
7
HARDWARE CONFIGURATION/ INSTALLATION
7
SOFTWARE CONFIGURATION/ INSTALLATION
10
SYSTEM PERIPHERALS -PRINTERS-
DISKS AND TAPES
BANKING ENVIRON.
11
APPLICATION PACKAGES
7
SYSTEM SOFTWARE MAINTENANCE

Title: DMS - DATA MANAGEMENT SYSTEM - User Guide

Newsletter Code: 4002362 L
Date: 22.03.84
Publication Code: 4002360 F(0)
Previous Newsletters: 4002361 G

This Newsletter provides updated pages for the subject publication.

The last level completed on the attached form, Updating Status, indicates the pages to be added, removed, or replaced, the number of pages included, and the Newsletter Code. Pages marked with an asterisk should be removed from the publication. The form should be filed at the back of the publication as a permanent record of amended pages.

Each amended page is identified by the Newsletter Code shown above.
Amended pages remain valid unless otherwise noted in a subsequent Newsletter.
Modifications to text, figures, or tables are indicated by a vertical bar in the outside margin next to the change.

Summary of Amendments:

Some function key meanings have been modified and DMS compatible screen format has been added.

Ing. C. Olivetti & C., S.p.A. - Direzione Documentazione - Via Jervis 77 10015 IVREA (TO) Italy

Printed in Italy

olivetti

UPDATING STATUS

LEVEL	DATE	UPDATED PAGES	PAGES	CODE
0	7.01.83		140	4002360 F
1	7.12.83	Preface, i, ii, iii, 1.0.1, 2.1.1, 2.1.3 ÷ 2.1.6, 2.2.1, 2.3.1, 2.3.2, 3.0.1, 3.0.2, 3.0.5, 3.0.6. 4.0.4, 5.0.1 ÷ 5.0.4, 5.0.7, 5.0.17 ÷ 5.0.27, 6.0.1 ÷ 6.0.7, 7.0.1 ÷ 7.0.3 CREATE.3, CREATE.4, DISPLAY.3, EXIT.1, EXIT.2 EXPUNGE.1, EXPUNGE.2, MODIFY.3, MODIFY.4, MOVE.3, OPEN.3, POSITION.4, PRINT.2, PRINT.3, PRINT.4, REPEAT.1 ÷ REPEAT.3, REPORT.2 ÷ REPORT.9 , RETURN.1, 9.0.1 ÷ 9.0.6, A.0.3, B.0.2, B.0.8	78	4002361 G
2	22.3.84	2.0.1, 2.3.5, 2.3.6, CREATE.2, MODIFY.2	5	4002362 L

Pages marked * must be suppressed

M30 M40

DMS - DATA MANAGEMENT SYSTEM

User Guide

olivetti L1

PREFACE

This manual describes the operating standards valid in the DMS environment and the programming by using DML (Data Manipulation Language).

This manual is principally intended for the application user and also for the person responsible for the installation of the DMS, i.e. the data administrator.

SUMMARY

This manual describes the following subjects:

- DMS operating environment
- DMS language elements
- The command file
- The procedures
- Error handling
- DML commands

This manual also includes:

- An example of DML programming
- Interactive specification of commands

The appendix contains a list and description of the error messages.

REFERENCES:

Read first...

Introduction to MOS
Code 4002130 G (vol. 2)

DMS - Features and Functions
Code 4002610 Q (vol. 2)

For further information, read...

DMS - Data Definition Language
User Guide
Code 4004570 D

MOS - EDITOR Reference Manual
Code 4002440 P (vol. 6)

MOS SHELL - Commands Reference Manual
Code 4002770 Q (vol. 3)

Glossary/Glossario
Code 4002140 H (vol. 1)

First Edition: January, 1983
Release 1.0

Second Edition: December, 1983
Release 2.0

PUBLICATION ISSUED BY:

Ing. C. Olivetti & C., S.p.A.
Direzione Documentazione
77, Via Jervis - 10015 IVREA (Italy)

4002361 G

1. INTRODUCTION

This manual describes the structure and rules for the DMS operating environment and the DML language commands.
The manual comprises the following chapters:

1. INTRODUCTION
2. THE DMS SESSION: OPERATING RULES
3. THE COMMAND FILE
4. ERROR HANDLING
5. DMS LANGUAGE ELEMENTS
6. PROCEDURES
7. DML COMMANDS
8. AN EXAMPLE OF A DMS APPLICATION
9. THE DMS.HELP FILE

Chapters 1 - 6 illustrate the structure and rules for the DMS operating environment.
Chapter 7 contains the DML language commands.
Chapter 8 gives a detailed example of a command file application.
Chapter 9 provides information on the DMS.HELP file, that can be associated to the DMS for interactive use of the main DML commands.

Appendices A and B describe, respectively, the error codes and the operator messages which may arise in DMS environment.

As it will be referred to frequently in this publication, it is assumed that the reader is already acquainted with the manual "DMS - Data Management System - Features and Functions".

However, a short overview is given of the concepts of the above manual that refer explicitly to the user programming environment.

Current Collection

The "current collection" is defined as the last collection opened during a DMS session.

Current Record

The record pointed to by the collection cursor at any instance is referred to as the 'current record'. When a collection has just been opened, the current record is the first in the file.
The DMS reads the current record into its own memory area and makes it available to the user.

Connecting to a Data Dictionary

At the beginning of each session, the DMS is automatically connected to the standard Data Dictionary.
Subsequently, the user can connect to other non-standard Data Dictionaries to refer to the collections they contain.
The standard and non standard Data Dictionaries must have been generated beforehand by the data administrator.
It should be remembered that:

- only one Data Dictionary at a time can be connected to the DMS

- if the end user connects to several Data Dictionaries during a DMS session, he can still access the collections belonging to these Data Dictionaries even after they have been disconnected.
  In fact, the information contained in the COLLECTION, VIEW and FILE DESCRIPTOR descriptors is read into the DMS memory when the collection is opened and remains available until the collection is closed.

2. THE DMS SESSION: OPERATING RULES

This chapter describes how to open, execute and close a DMS session.
The description does not refer to a specific job to be performed but to the DMS environment as such.
The following are explained:

- connection to the DML Interpreter

- entry and execution of DML commands

- screen handling

- exit from DMS.

This chapter begins illustrating the "function keys" section of the keyboard:the functions represented are those used by the DMS and they are described in the present chapter.
The alphanumeric section of the keyboard is not given as it changes according to the country where it is used.
Note, however, that the symbols:
$, £
may be substituted by the corresponding symbols
¤, #
depending on the type of keyboard provided.
In this manual, the
$ and £
symbols have been used.

Screen format

The screens must have the 25x80 format for DMS. This can be changed to the 13x40 format only when creating the Data Dictionary.

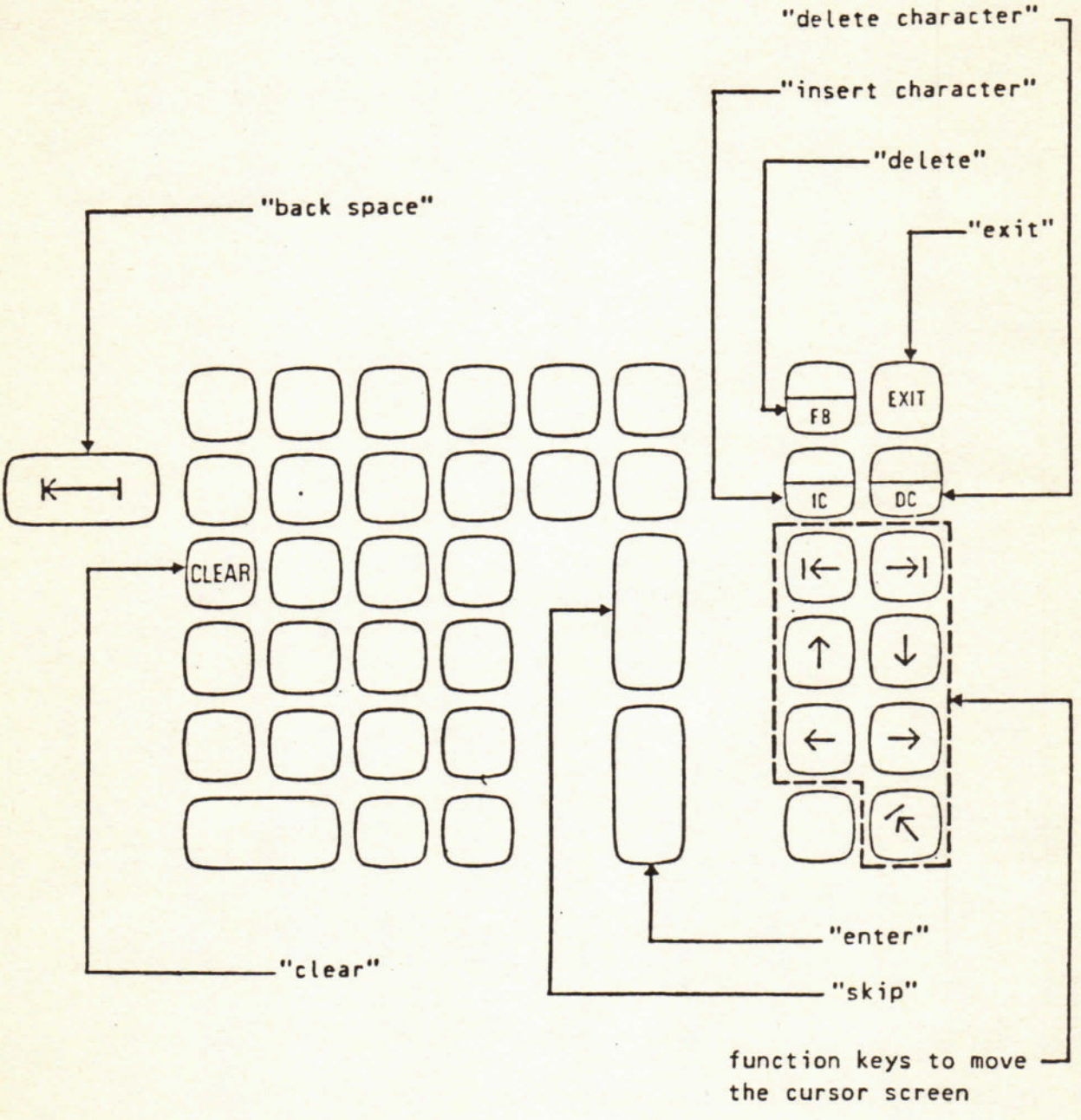


Fig. 2.1 - The Function Keys Section.

CONNECTION TO THE DML INTERPRETER

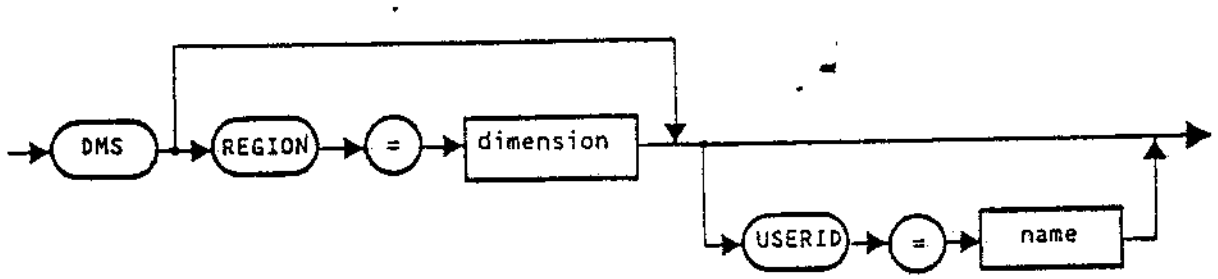
Connection to the DML Interpreter can be done in two ways:

- as an option on the initial work station menu

- in the SHELL environment.
  Access from SHELL is made by using the command:
  DMS

  This connection mode is described in the following pages.

When this command is entered under SHELL, connection is made to the DML (Data Manipulation Language) interpreter
At the same time the user can specify the size of the memory area.
The enabled user identifier can either be entered later or when calling DMS.



where:

REGION — Is the size of the memory area to be made available. It will contain the commands, which are either keyed-in by the user or entered by using a command file.

dimension — specifies the dimension of REGION in bytes. The maximum number is 64 Kbytes.

If this is not specified, 2500 bytes is assumed by default.

USERID This is for directly entering a user identifier enabled to access the standard Data Dictionary, when connecting to the DML.

name specifies the user identifier that can be of the data administrator or user application type.

Characteristics When the dimensions of the memory area are insufficient to contain the sequence of commands, the message:

PROGRAM TOO LONG

is displayed.

When input is redirected (see the paragraph "Redirection of Input and Output" in the chapter "THE COMMAND FILE") the USERID and the REGION must be supplied when DMS is called.

DMS USERID=name < input_file

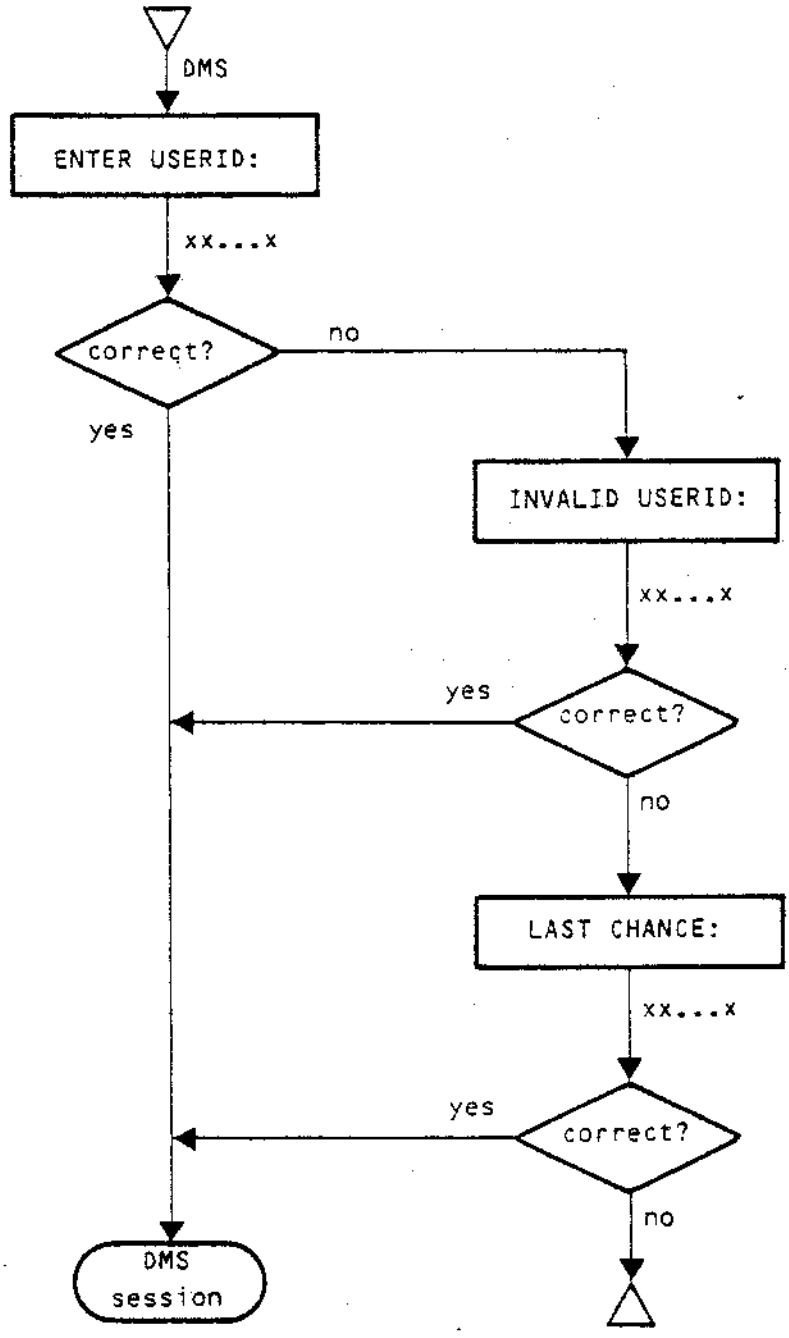
or

DMS REGION=dimension USERID=name < input_file

(where input file is the name of the file to be executed)

Interactive Input of User Identifier

When the command DMS has been keyed-in without further specifications, the request for the user identifier is made as shown in the following figure (see also the next page).



ENTER USERID :	This message is displayed after DMS has been keyed-in and connection has been made to the DML interpreter.
XX...X	The user must key-in an identifier (which is not displayed) from the USERID descriptor of the standard Data Dictionary. If the identifier is correct the DMS session begins, if it is not, the following message is displayed.

INVALID USERID :	This message is displayed when the identifier that has been keyed-in is not in the USERID descriptor.
XX...X	A correct identifier must be keyed-in (which is not displayed).

LAST CHANCE :	This message is displayed when the second identifier is incorrect. The user has one more chance to input a correct one, otherwise the system will disconnect from DMS and return to SHELL.
XX...X	A correct identifier must be keyed-in (which is not displayed).

ENTRY AND EXECUTION OF DML COMMANDS

When entering and executing DML commands, the user should remember that:

- the command is keyed-in, displayed and then loaded into the DMS memory.
  The "DML COMMANDS" chapter explains the syntax and individual parameters of each command.

- the elements of a command must be separated by at least one blank. If they are in list form, they are separated by the ',' character.
  A single command or a set of commands may occupy one or more lines on the screen.

- several consecutive commands must be separated by the ';' character. This is not mandatory where only one command is entered and executed.

- using the MOS Editor, a set of commands can be entered and stored in a file and executed later. In this case, comments may be inserted in the commands. (Refer to the chapter "THE COMMAND FILE").

- up to 200 characters can be entered before pressing the /ENTER/ key (see below).
  This restriction is due to the size of the keyboard buffer. An acoustic signal alerts the user when the character entered fills the available area.

- When /ENTER/ key is pressed, the command or set of commands entered is executed immediately.
  A set of commands is executed in order of entry.

- The / ⟵ / key represents the back-space function which:
  moves the cursor one character to the left, then the character that the cursor is pointing to is deleted.

  If the cursor is pointing to the first position of the input area, an acoustic signal alerts the user and pressing of the back-space key has no effect.
  For example, with the following input and the cursor pointing to 'D':

  A B C D E F G H

  after pressing the back-space key:

  A B _ D E F G H

  will be displayed.
  If the back-space key is pressed again, the input area is as follows:

  A _ D E F G H

  and so on.

HANDLING OF THE SCREEN

This paragraph describes from an operational point of view how to enter and modify the values using the following commands:

CREATE, MODIFY and DISPLAY

Note that the record values are not displayed if the CREATE and MODIFY commands include the option:

USING.....statements.....ENDUSING

and DISPLAY command has not been specified between the statements. (Refer to CREATE and MODIFY commands in the chapter "DML COMMANDS").

The following are explained below:

- checking of correct input length
- the function keys used for editing.

Checking of Correct Input Length

When a field value is entered or modified, the DMS checks the length of the field.
The following cases apply:

- string and numeric display type fields (internal representation in ASCII code).
  An acoustic signal alerts the user when the last accepted character has been entered.
- integer numeric fields, single and double precision fields.
  A check is made on the entered value at the field closure, according to the following values:

. integer numeric fields:
The order of magnitude of the number depends on how many 9's are declared in the PICTURE clause.
See the table corresponding to the USAGE format clause, specifying COMP, in the VIEW command. (This refers to "DMS - Data Definition Language - User Guide").

. single precision fields:
4 bytes representation; the value range is from 10^{-37} to 10^{+38}.
See the USAGE format clause, specifying COMP-1, in the VIEW command.

. double precision fields:
8 bytes representation; the value range is from 10^{-307} to 10^{+308}.
See the USAGE format clause, specifying COMP-2, in the VIEW command.

- packed numeric fields.
The amount of digits accepted depends on how many 9's are declared in the PICTURE clause. See the USAGE format clause, specifying COMP-3, in the VIEW command.

Function Keys

The function keys described below may be used to edit:

- characters
- fields
- records

characters

/←—/ Moves the cursor to the previous position of the input area.
If it is pointing to the first position of the area, the cursor does not move and an acoustic signal alerts the user.

For example, with the following input and the cursor pointing to 'D':

A B C D E F G H

after pressing the key:

A B _ D E F G H

will be displayed.

If it is pressed again, the input area is as follows:

A _ D E F G H

and so on.

fields

/ENTER/ or /↓/ Moves the cursor to the first position of the next field.
If the cursor is pointing to the last field of the page, then:

- if there is another page, this is automatically displayed

- if there are no more pages, execution of the current command is closed.
  The DMS will execute the next command, if any; otherwise, the DMS prompt is displayed.

  Note that when the ALL option is specified in the CREATE and MODIFY commands, the 'next page' also includes the first page of any records after the current record.

/ ↑ / Moves the cursor the first position of the preceding field.
If the current field is the first of the page, this key has the same effect as the /|← / key.

/ ↖ / Moves the cursor to the first position of the first field of the screen page.
If the current field is the first of the page, this key has the same effect as the /|← / key.

records

/EXIT/ Pressing of this key:

terminates operations on the current record. If during the execution of a CREATE or MODIFY command the current record will be stored on disk.
If the option ALL was specified, the first page of the next record, if any, will be displayed.

/SKIP/ Pressing this key:

terminates operations on the current record. If during the execution of a CREATE or MODIFY command, the current record will not be stored on disk.
If the option ALL was specified, the first page of the next record, if any will be displayed.

/CONTROL//EXIT/ Pressing this key:

terminates operations on the current record. If during the execution of a CREATE or MODIFY command, the current record will not be stored on disk.
The command will be interrupt even if the ALL option had been specified.

3. THE COMMAND FILE

This chapter describes the command file, how it is created and used.

Redirection of Input and Output and Completion Code are also illustrated.

Definition

A command file is a disk file containing a set of DMS commands.

All DDL and DML commands can be stored on a command file, but this can only contain either one or the other at any one time.
This is because when the stored commands are executed, only one of the interpreters (DDL or DML) is in memory.

The commands in a command file are executed in the order in which they were recorded.

Creation

Command files are created in SHELL environment, connecting to the Editor by keying in:

EDITOR file_name

For further details of the Editor program, refer to:

MOS - EDITOR Reference Manual

to which reference is made here.

file name

Assignment of the command file name is as specified for the Editor environment.
This file name (always < 12 characters) can be given as follows:

- name (which is appended to the current directory)
- partial path name ending with the file name
- complete path name ending with the file name.

As can be seen from the above diagram, during nested execution of several command files, the instructions of all the files being executed are in the memory at the same time.
The only restriction on the nested execution of several command files is the size of the available DMS memory area to store commands (REGION parameter).
Note that:

- when a group of commands is stored on disk; any sequences of blanks are reduced to a single blank character
- any comments inserted between command file instructions are stored on disk.
  When the instructions are read and transferred into the memory, the comments are reduced to a single blank character.
- the memory area occupied by a command file can be calculated adding together:
  - the number of characters used to write the commands
  - a blank for each set of blanks in the text
  - a blank for each comment in the text.

Redirection of Input and Output

Apart from being activated by the EXEC command, a DDL or DML command file can also be run directly under SHELL environment using the following command which starts DMS and then executes the specified command file.

```
ADMS USERID=asterix <file_name
```
(for a DDL command file).

```
DMS USERID=obelix <file_name
```
(for a DML command file).

where file name is the name of the command file to be executed.

The last command in the command file must be EXIT.

In addition to being displayed or printed, output from file command execution can also be stored on another file, written thus:

```
DMS USERID=idefix <file_name_1 >file_name_2
```

where file name 2 (output file) is created automatically if it does not exist already.

The file that is to be executed (filename 1) must neither contain the DISPLAY command nor the CREATE and MODIFY commands used in interactive mode; they can therefore only be used when associated to the USING parameter.

Completion Code

Values can be passed from the DMS environment to the SHELL environment, via the EXIT command (only for DML commands).

The value associated to EXIT is passed to the STATUS variable in the SHELL environment, via the "expression" parameter.

This can be useful for SHELL procedures that contain DMS calls.

Example.

A normal SHELL procedure is for executing command file, file1, and, if there are no errors, also command file2.

```
DMS USERID=asterix <file1 >file_out1;
IF %STATUS = 0
THEN DMS USERID=asterix <file2 >file_out2;
```

When a command file is executed like this commands are interpreted line by line, so those like IF...ENDIF or REPEAT...END cannot be written on more than one line.

ERROR Option

When an error occurs, the DMS displays:

```
**...error message ...(nn)
NOW YOU CAN:

S - SKIP THE COMMAND
R - RETRY THE COMMAND
P - DMS PROMPT
A - ABORT THE CF
E - EXIT FROM DMS

PLEASE ENTER OPTION:
```

Where:

- nn
  is the numeric code identifying the error. Can be tested in a next instruction (see below)
- S - SKIP THE COMMAND
  this option is selected entering S
  execution of the command that caused the error will be abandoned and the next command will be executed.
  If there are no more commands the DMS prompt is displayed; DMS is now waiting for further commands
- R - RETRY THE COMMAND
  this option is selected entering R
  the command is retried.
- P - DMS PROMPT
  this option is selected entering P
  execution of the command that caused the error is abandoned and any subsequent commands are not executed.
  The DMS prompt is displayed; DMS is now waiting for further commands.
- A - ABORT THE CF
  this option is selected entering A
  it can be selected only when the error has occurred during execution of a command in a command file.

Execution of the command file that caused the error is abandoned and the next command is executed. If there are no more commands the DMS prompt is displayed; DMS is now waiting for further commands.

- E - EXIT FROM DMS
  this option is selected entering E
  execution of the command that caused the error is abandoned, without executing any subsequent commands. The DMS session ends and the SHELL prompt is displayed.
  Note that entry of E has the same effect as execution of the EXIT command (see the chapter "DML COMMANDS").

Option I

When the message error and the operation selection menu are displayed, option I can be specified in addition to those indicated already.
The command that caused the error, and the position of the error itself are indicated.

If the SHOW command is keyed-in incorrectly:

```
SHOP a
```

the following message is displayed:

```
**SHOP: COMMAND NOT RECOGNISED (1)
```

followed by the option menu.
If the option 1 is entered the following appears:

```
SHOP a
   $
```

where $ indicates where the error occurred.

NOERROR Option

If this option has been selected, no information is displayed in the case of an error.
Execution of the command is abandoned and the next command is executed.
If there are no more commands the DMS prompt is displayed; DMS is now waiting for further commands.

5. DMS LANGUAGE ELEMENTS

This chapter explains the semantics of the terms (elements) used to write DMS commands.
Each element is described in a separate paragraph.

KEYWORD

Keywords must be written as specified in the general command structure.
DMS keywords may be abbreviated to the first three letters.
Where several keywords start with the same three letters, the fourth letter must be specified to avoid confusion.

Example.

The names of the following DML commands are keywords:

- OPEN can be shortened to OPE
- MODIFY can be shortened to MOD
- DISPLAY can be shortened to DIS
- REPORT can be shortened to REPO

The REPEAT command is not covered by this rule, as it can never be shortened.

NAME

A name is a sequence of characters from the following set:

- A - Z
- a - z
- Ø - 9
- _ (underscore)
- " (quotes) used to accentuate

Each name:

- must start with an alphabetic character, either upper or lower case.
- may be of any length but only the leftmost 30 characters are significant.

Note

The names of DMS descriptors may not exceed 12 characters.
This limit is imposed by the file system value (an element of a pathname may not exceed 12 characters).

a	b	c	d	a	b	c	d	a	b	c	a	b	c
0	00	0000 0000	NUL	64	40	0100 0000	@	128	80	1000 0000	192	C0	1100 0000
1	01	0000 0001	SOH	65	41	0100 0001	A	129	81	1000 0001	193	C1	1100 0001
2	02	0000 0010	STX	66	42	0100 0010	B	130	82	1000 0010	194	C2	1100 0010
3	03	0000 0011	ETX	67	43	0100 0011	C	131	83	1000 0011	195	C3	1100 0011
4	04	0000 0100	EOT	68	44	0100 0100	D	132	84	1000 0100	196	C4	1100 0100
5	05	0000 0101	ENQ	69	45	0100 0101	E	133	85	1000 0101	197	C5	1100 0101
6	06	0000 0110	ACK	70	46	0100 0110	F	134	86	1000 0110	198	C6	1100 0110
7	07	0000 0111	BEL	71	47	0100 0111	G	135	87	1000 0111	199	C7	1100 0111
8	08	0000 1000	BS	72	48	0100 1000	H	136	88	1000 1000	200	C8	1100 1000
9	09	0000 1001	HT	73	49	0100 1001	I	137	89	1000 1001	201	C9	1100 1001
10	0A	0000 1010	LF	74	4A	0100 1010	J	138	8A	1000 1010	202	CA	1100 1010
11	0B	0000 1011	VT	75	4B	0100 1011	K	139	8B	1000 1011	203	CB	1100 1011
12	0C	0000 1100	FF	76	4C	0100 1100	L	140	8C	1000 1100	204	CC	1100 1100
13	0D	0000 1101	CR	77	4D	0100 1101	M	141	8D	1000 1101	205	CD	1100 1101
14	0E	0000 1110	SO	78	4E	0100 1110	N	142	8E	1000 1110	206	CE	1100 1110
15	0F	0000 1111	SI	79	4F	0100 1111	O	143	8F	1000 1111	207	CF	1100 1111
16	10	0001 0000	DLE	80	50	0101 0000	P	144	90	1001 0000	208	D0	1101 0000
17	11	0001 0001	DC1	81	51	0101 0001	Q	145	91	1001 0001	209	D1	1101 0001
18	12	0001 0010	DC2	82	52	0101 0010	R	146	92	1001 0010	210	D2	1101 0010
19	13	0001 0011	DC3	83	53	0101 0011	S	147	93	1001 0011	211	D3	1101 0011
20	14	0001 0100	DC4	84	54	0101 0100	T	148	94	1001 0100	212	D4	1101 0100
21	15	0001 0101	NAK	85	55	0101 0101	U	149	95	1001 0101	213	D5	1101 0101
22	16	0001 0110	SYN	86	56	0101 0110	V	150	96	1001 0110	214	D6	1101 0110
23	17	0001 0111	ETB	87	57	0101 0111	W	151	97	1001 0111	215	D7	1101 0111
24	18	0001 1000	CAN	88	58	0101 1000	X	152	98	1001 1000	216	D8	1101 1000
25	19	0001 1001	EM	89	59	0101 1001	Y	153	99	1001 1001	217	D9	1101 1001
26	1A	0001 1010	SUB	90	5A	0101 1010	Z	154	9A	1001 1010	218	DA	1101 1010
27	1B	0001 1011	ESC	91	5B	0101 1011	[	155	9B	1001 1011	219	DB	1101 1011
28	1C	0001 1100	FS	92	5C	0101 1100	\	156	9C	1001 1100	220	DC	1101 1100
29	1D	0001 1101	GS	93	5D	0101 1101	]	157	9D	1001 1101	221	DD	1101 1101
30	1E	0001 1110	RS	94	5E	0101 1110	↑	158	9E	1001 1110	222	DE	1101 1110
31	1F	0001 1111	US	95	5F	0101 1111	—	159	9F	1001 1111	223	DF	1101 1111
32	20	0010 0000	SPACE	96	60	0110 0000	`	160	A0	1010 0000	224	E0	1110 0000
33	21	0010 0001	!	97	61	0110 0001	a	161	A1	1010 0001	225	E1	1110 0001
34	22	0010 0010	"	98	62	0110 0010	b	162	A2	1010 0010	226	E2	1110 0010
35	23	0010 0011	#	99	63	0110 0011	c	163	A3	1010 0011	227	E3	1110 0011
36	24	0010 0100	$	100	64	0110 0100	d	164	A4	1010 0100	228	E4	1110 0100
37	25	0010 0101	%	101	65	0110 0101	e	165	A5	1010 0101	229	E5	1110 0101
38	26	0010 0110	&	102	66	0110 0110	f	166	A6	1010 0110	230	E6	1110 0110
39	27	0010 0111	'	103	67	0110 0111	g	167	A7	1010 0111	231	E7	1110 0111
40	28	0010 1000	(	104	68	0110 1000	h	168	A8	1010 1000	232	E8	1110 1000
41	29	0010 1001	)	105	69	0110 1001	i	169	A9	1010 1001	233	E9	1110 1001
42	2A	0010 1010	*	106	6A	0110 1010	j	170	AA	1010 1010	234	EA	1110 1010
43	2B	0010 1011	+	107	6B	0110 1011	k	171	AB	1010 1011	235	EB	1110 1011
44	2C	0010 1100	,	108	6C	0110 1100	l	172	AC	1010 1100	236	EC	1110 1100
45	2D	0010 1101	-	109	6D	0110 1101	m	173	AD	1010 1101	237	ED	1110 1101
46	2E	0010 1110	.	110	6E	0110 1110	n	174	AE	1010 1110	238	EE	1110 1110
47	2F	0010 1111	/	111	6F	0110 1111	o	175	AF	1010 1111	239	EF	1110 1111
48	30	0011 0000	0	112	70	0111 0000	p	176	B0	1011 0000	240	F0	1111 0000
49	31	0011 0001	1	113	71	0111 0001	q	177	B1	1011 0001	241	F1	1111 0001
50	32	0011 0010	2	114	72	0111 0010	r	178	B2	1011 0010	242	F2	1111 0010
51	33	0011 0011	3	115	73	0111 0011	s	179	B3	1011 0011	243	F3	1111 0011
52	34	0011 0100	4	116	74	0111 0100	t	180	B4	1011 0100	244	F4	1111 0100
53	35	0011 0101	5	117	75	0111 0101	u	181	B5	1011 0101	245	F5	1111 0101
54	36	0011 0110	6	118	76	0111 0110	v	182	B6	1011 0110	246	F6	1111 0110
55	37	0011 0111	7	119	77	0111 0111	w	183	B7	1011 0111	247	F7	1111 0111
56	38	0011 1000	8	120	78	0111 1000	x	184	B8	1011 1000	248	F8	1111 1000
57	39	0011 1001	9	121	79	0111 1001	y	185	B9	1011 1001	249	F9	1111 1001
58	3A	0011 1010	:	122	7A	0111 1010	z	186	BA	1011 1010	250	FA	1111 1010
59	3B	0011 1011	;	123	7B	0111 1011	{	187	BB	1011 1011	251	FB	1111 1011
60	3C	0011 1100	<	124	7C	0111 1100	|	188	BC	1011 1100	252	FC	1111 1100
61	3D	0011 1101	=	125	7D	0111 1101	}	189	BD	1011 1101	253	FD	1111 1101
62	3E	0011 1110	>	126	7E	0111 1110	~	190	BE	1011 1110	254	FE	1111 1110
63	3F	0011 1111	?	127	7F	0111 1111	DEL	191	BF	1011 1111	255	FF	1111 1111

This table shows the equivalence of the ASCII code (d) in the representation:

- decimal (a)
- hexadecimal (b)
- 8 bit Binary (c)

Note The squared characters are different on the national keyboards.

The field containing the value 'BLACK' can be addressed as:

COLLECTION1:A.B.C or COLLECTION1:B.C
or COLLECTION1:C

The field containing the value 'GREEN' can be addressed as:

COLLECTION1:A.E.C or COLLECTION1:E.C

- if the identifier indicates a group of fields and not just a single field, the contents of all the fields of the group are addressed.

  COLLECTION1:A.E

  refers to fields C and D of group E.
  The contents of these fields are considered as chained: GREENYELLOW

  or separate: GREEN YELLOW

  according to the DML command being executed.

- when an identifier indicates a single element of a table-like structure, it is followed by the number (between brackets) that identifies the element.

  In a table defined as follows:

```
01 TABLE.
  02 ELEMENT PIC  XX OCCURS 50 TIMES.
```

  TABLE.ELEMENT(17) identifies the 17th element of the table.

  It can also be addressed using a previously allocated and defined variable.

```
ALOC x;
MOVE 17 to %x
```

  Then the following identifier can be used:

  TABLE.ELEMENT(x)

  which still indicates the 17the element of the table.

NUMBER

Indicates an integer or decimal number with or without sign.
In DMS the decimal point '.' is used.
A number without sign is considered as positive.

There cannot be more than 15 digits in a number including the decimal places.
In the case of an overflow, the 15th to 30th digits are converted into significant zeroes to preserve the order of magnitude of the number.
In this case, the DMS automatically uses exponential notation (see below).
For example, the following number:

- 123456789012345999

  is converted into

- 123456789012345E+3

Note that when numeric fields are entered, exponential notation may be used according to accepted conventions for expressing single or double precision numbers.
For example, 100000 can be expressed as:

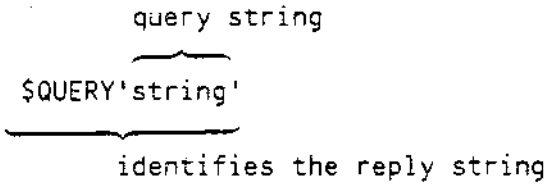
- in single precision:
  1E+5 or 100E+3 etc.

- in double precision:
  1D+5 or 100D+3 etc.

$CR	Positions the cursor on column 1 of the next line. If the cursor is on the last line, the screen page is scrolled up by one line and: . the last line of the screen is now accessible . the first line of the screen is lost.
$BEEP	Produces an acoustic signal.
$AT(expression1: expression2)	This symbol defines the cursor position. Expression1 and expression2 are two DMS language EXPRESSION elements indicating: - expression1 = the video line - expression2 = the video column The symbol is normally used with the SHOW and OUTPUT commands, e.g. the command: OUTPUT $AT(1:20), 'DAY"S DATE' displays: DAY'S DATE starting from the 1st line and the 20th column. Note that when expression1 or expression2 contains a field name starting from the collection name, e.g. coll01:space.field05 or only a field name, e.g. field05 to avoid incorrect use of character ':', the colon that separates expression1 and expression2, must be preceded by a blank, e.g.: OUTPUT $AT(coll01:space.field05:coll01:space.field07) and OUTPUT $AT(field05:field07)

$QUERY'string'

Identifies the string entered by the operator in reply to the query provided after $QUERY.
The reply to $QUERY'string' is considered a string type entity.
For example:



For example, the following comand:

MOVE $QUERY'DAY''S DATE' to %DATE
will:

- display the 'DAY ' S DATE' string
- the date entered by the operator is assigned to the DATE variable.

$FORMAT'mask'

Edits the expression that precedes this symbol in conformity with the mask provided.
The mask characters may be any displayable character and are used as follows:

- '£' indicates a position that will be occupied by a character of the expression being edited
- '.' indicates the position of the decimal point in the edited expression
- all the other characters will be reproduced as such in the edited expression.

The expression is edited as follows:

string value

The expression to be edited is a string value.

The characters to be edited are inserted in order from left to right in the mask positions containing the '£' character.
If the number of £ characters differs from the number of characters to be edited:
any extra '£' characters will be replaced with a blank
if there are fewer '£' characters, the extra characters in the expression to be edited will be lost.

Examples.

```
SHOW '820615' $FORMAT'YEAR ££ / MONTH ££ / DAY ££'
is edited as
YEAR 82 / MONTH 06 / DAY15

SHOW '163' $FORMAT'SERIAL NUMBER : £££££'
is edited as:
SERIAL NUMBER : 163ØØ

SHOW 'AZ9874' $FORMAT'CODE: ££££'
is edited as:
CODE: AZ98
```

numeric value

The expression to be edited is a numeric value.
There are two possibilities:

- The mask does not contain the decimal point.
  In this case:

  . the fractional part of the number is lost.
  The digits of the integer part are inserted in order from right to left in the mask positions containing '£'.

  . extra '£' characters are replaced with blanks; if there are not enough '£' characters, the entire number will be lost and, in the edited expression, the '£' characters are replaced with '*'.

Examples.

. SHOW 1728.5 $FORMAT'£££££'
  is edited as
  ØØ1728

. SHOW 1728.5 $FORMAT'£££'
  is edited as
  ***

- the mask contains the decimal point.
  In this case:

  . the digits of the integer are handled as above (i.e. mask without decimal point);
    editing is from right to left starting from the decimal point

  . the fractional part is inserted in order from left to right in the mask positions containing '£' starting from the decimal point.
    Any extra '£' characters are replaced with Ø.
    If there are not enough '£' characters, the extra digits will be lost (no rounding off).

Examples.

. SHOW 1728.5 $FORMAT'£££££.£££'
  is edited as
  Ø1728.5ØØ

. SHOW 1728.594 $FORMAT'£££££.£'
  is edited as:
  Ø1728.5

. SHOW 1728.5 $FORMAT'£££.£'
  is edited as
  ****

$I

This symbol corresponds to a variable that can only be associated to the REPEAT command.
This variable is set at 1 when a REPEAT cycle is executed and it keeps count of the times this is repeated. This special sign can be assigned to each REPEAT command, so that if several REPEAT cycles are nested, various $I symbols can be used; each one of these can be used to control its relative cycle.

The following is an example of $I associated to REPEAT.

```
REPEAT 3 TIMES
  SHOW $I;
END
```

The following sequence will then be displayed:

```
1
2
3
```

In the next example $I is associated to nested REPEAT cycles:

```
REPEAT 3 TIMES
  SHOW $I;
  REPEAT $I TIMES
    SHOW 'ØØ', $I;
  END
END
```

The following sequence is displayed:

```
1
  1
2
  1
  2
3
  1
  2
  3
```

$V(expr.1:expr.2) This symbol enables use of visual attributes, i.e. the special characters and signs that are for emphasising what is shown on the screen.
The type of visual operation and the screen area in which it is executed are indicated by expression 1 and expression 2 (see EXPRESSION).

Expression 1 must contain a number that qualifies visual attributes.
The elementary visual attributes are shown in the following list with their identifying numbers in the next column.

VISUAL ATTRIBUTES	IDENTIFIER
Overscore	1
Underscore	2
Left margin	4
Right margin	8
Blinking	16
Highlight	32
Reverse	64
No-echo	96
Visual attribute delete	0
Character delete	-1

The numbers can be added together so as to have more than one visual effect at a time, (only 'highlight' '32' and 'reverse' '64' cannot be used at the same time).
For example: the number 3 would mean overscore plus underscore; 15 would produce a small rectangle (overscore plus underscore plus left and right margin), etc.

Expression 2 shows the screen area in which the visual attribute specified in expression 1 operates. It must contain a number showing in how many columns to the right of the current cursor position, the visual attribute is effective.

The visual attributes start from the current cursor position but they do not move it.

Examples.

```
SHOW $FF;
OUT $AT(2:4Ø),$V(2:21), 'GENERAL HEADING'
```

the result of this, after the screen has been cleared, will be:

GENERAL HEADING

This will be shown on the second line, starting from column 4Ø and will be underlined as defined by the visual attribute.

The following example command file shows the use of visual attributes used to emphasise a menu (see also the figure illustrating the example).

```
ALOC OPT;
REPEAT
  SHOW  $FF,
        $V(1:8Ø),
        $AT(24:1),$V(1:8Ø),
        $AT(1:23),'OPTION LIST',
        $AT(1:23),$V(2:2Ø),
        $AT(3:1),$V(1:80),
        $AT(1Ø:1),$V(2:8Ø);
  OUT $AT(24:33);
  IF $ERR THEN OUT '**ERROR**' ENDIF;
  REPEAT 7 TIMES
    OUT $AT($1+3:39),$V(4:1)
  END;
  OUT $AT(3:2),'POSSIBLE OPTION',$CR,
      'Ø - END', $CR,
      '1 - OPTION 1', $CR,
      '2 - OPTION 2', $CR,
      '3 - OPTION 3', $CR,
      '4 - OPTION 4', $AT(4:4Ø),
      '5 - OPTION 5', $AT(5:4Ø),
      '6 - OPTION 6', $AT(6:4Ø),
      '7 - OPTION 7', $AT(7:4Ø),
  $AT(1Ø:4Ø),'KEY-IN OPTION:';
  MOVE $QUERY 'ØØ' TO %OPT;
    IF %OPT = 'Ø' THEN BREAK ENDIF;
    IF %OPT = '1' THEN EXEC cfØ1 ENDIF;
    IF %OPT = '2' THEN EXEC cfØ2 ENDIF;
    IF %OPT = '3' THEN EXEC cfØ3 ENDIF;
    IF %OPT = '4' THEN EXEC cfØ4 ENDIF;
    IF %OPT = '5' THEN EXEC cfØ5 ENDIF;
    IF %OPT = '6' THEN EXEC cfØ6 ENDIF;
    IF %OPT = '7' THEN EXEC cfØ7 ENDIF;
END
. . . . . .
```

where cfØ1, cfØ2, etc. cause execution of specific command files, according to which option is keyed-in.

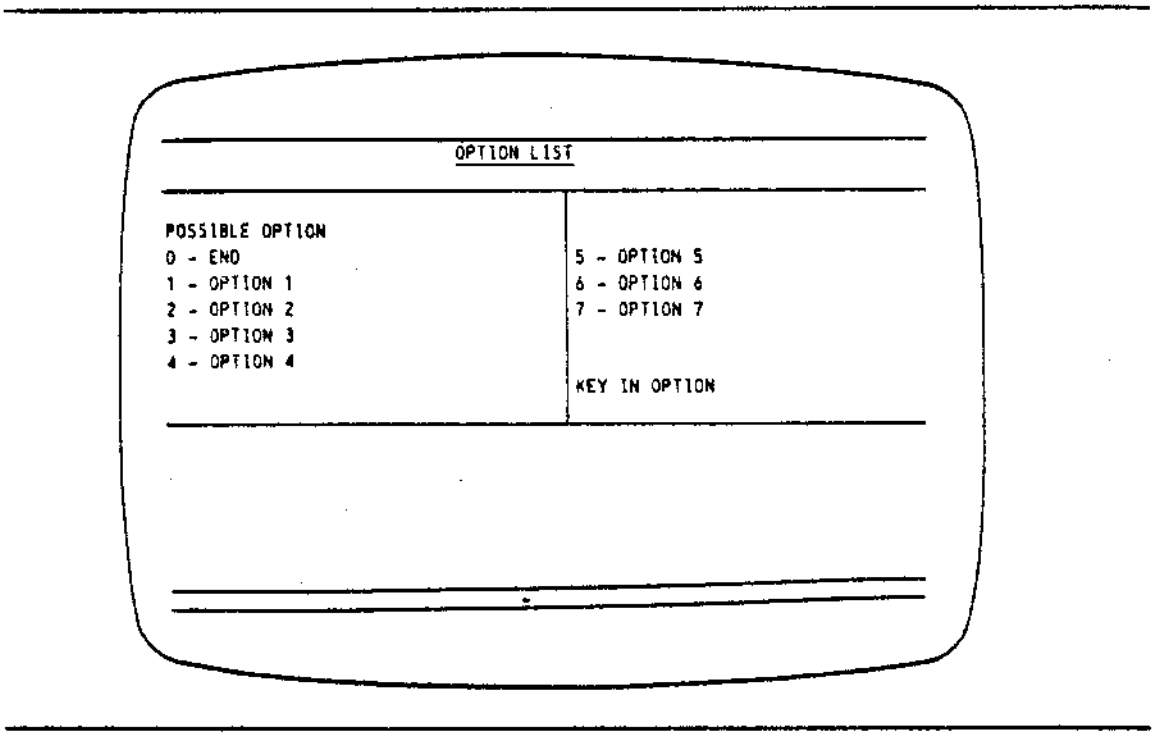


Fig. 5. 1 - Example of Visual Attributes Application

Note that when expression1 or expression2 contains a field name, which starts from the name of the collection, e.g.:

collA:vis.field1

or only a field name, e.g.

field1

to avoid incorrect use of the character ':', the colon that separates expression1 from expression2 must be preceded by a blank, e.g.:

OUT $V(collA:vis.field1Ø:collA:vis.field3)

and

OUT $V(field1Ø:field5)

colour video

If a colour video is being used, the values assumed by expression1 can be as follows (these cannot be summed together).

VISUAL ATTRIBUTES	IDENTIFIER
Overscore	1
Underscore	2
Left margin	4
Right margin	8
Red blinking	16
White	32
White blinking	48
Yellow in reverse	64
Yellow in reverse blinking	80
No-echo	96
Magenta in reverse	112
Yellow	128
Red	144
Light blue	160
Green blinking	176
Green in reverse	192
Red in reverse	208
Blue	224
Magenta	240
Visual attribute delete	0
Character delete	-1

EXPRESSION

An expression is either a:

- NAME
- STRING
- IDENTIFIER
- NUMBER
- VARIABLE
- SPECIAL SYMBOL

singly or joined by arithmetic and/or logical operators.

Given below are some examples of expressions:

- 'THE CUSTOMER CODE IS'
- COLLECTION1:a.b.c
- 13.6
- %VAR
- 153.4 $FORMAT 'UNIT PRICE: ££££.£'
- (COLLECTIONA:f.g + 100)* 150/100
- Collection17:h >= Collection17:m

When an expression is referred to it is always evalued and the result is either a numeric or string value.

6. THE PROCEDURE

This chapter defines a procedure and describes how it is created and used.

Definition

A procedure is a sequence of DML commands for executing generalised operations.

allocation

Procedures are created by the data administrator and they are stored in the Data Dictionary.
All the procedures are written in DDL environment and executed in DML environment.
All DML commands can be part of a procedure.

use

Procedures solve generalised problems, as far as applications are concerned, with the use of parameters. These parameters are indicated when the procedure is written, and when the user calls the procedure these are assigned actual values.

language extension

Procedures can be regarded as new commands composed of a set of commands and are often more efficient than command files.

Creation

All the procedures are written in DDL environment using the PROCEDURE command.
They are made up of:

- an identifying name
- a list of parameters
- a body composed of a sequence of DML commands

A very simple procedure could be written in the following way:

```
PROCEDURE double (a,b)
          SHOW a*2;
          SHOW b*2;
ENDPROC
```

Where "double" is the name identifying the procedure, "a" and "b" are the parameters, the two SHOW commands are the sequence which is the body of the procedure.

Name — Identifies the procedure in the Data Dictionary. This name is used to invoke the procedure for execution in DML environment.

Parameters — Parameters are variables whose value is initially defined by the user when the procedure is called. Values must be supplied for the number of parameters defined.

Parameters can be used in two different ways:

- directly (by reference)
- indirectly (by value)

direct assignment — Directly assigned parameters have their values assigned one by one during procedure execution.

indirect assignment — Indirectly assigned parameters (these are indicated by the parameter name preceded by ^) have their values assigned at the beginning of the execution procedure.

Example.

The following example shows the various meanings the parameters assume, according to assignment type.
In this example a collection containing 5 records has been opened; a field in the record, called field _a, contains in sequence the following values:

1
2
3
4
5

The following procedure contains a directly assigned parameter:

```
PROCEDURE show_d (p1)
   POSITION FIRST;
   REPEAT
     SHOW p1;
     POSITION NEXT;
     IF $EOC THEN BREAK
   END
ENDPROC
```

When the procedure is executed and field_a is indicated as the actual value, the following sequence is displayed.

```
1
2
3
4
5
```

On entry to the procedure, parameter p1 is replaced by the actual value, i.e. field_a, and is evaluated by the SHOW command during the REPEAT cycle.

If, on the other hand, the procedure has an indirectly assigned parameter:

```
PROCEDURE show_in (^p1)
  POSITION FIRST;
  REPEAT
    SHOW p1;
    POSITION NEXT;
    IF $EOC THEN BREAK
    ENDIF
  END
ENDPROC
```

When the same actual value is given again, the following sequence is displayed:

```
1
1
1
1
1
```

On entry to the procedure the parameter p1 is evaluated and is replaced by the value from field_a in the current record.
This value remains in force for the whole REPEAT cycle.

The two types of parameter may be used in the same procedure.

readability

Parameters can also be used to comment on the procedure or to make it more readable from a user point of view.
As far as the procedure is concerned, they have only a positional significance.
In the following example parameter p2 is for readability:

```
PROC open (p1,p2,p3)
   OPEN p1 MODE p3
ENDPROC
```

When the "open" procedure is executed under DML, parameter p2 could be replaced by an actual value used as an explanation, e.g. "collection" or "catalogue".

So if a collection called MUSIC is to be opened for modification, the following can be written:

```
open MUSIC collection MODIFY
```

or

```
open MUSIC catalogue MODIFY
```

Readability parameters must always be direct access.

Procedure body

The body of the procedure is made up of DML commands.
The RETURN command, in particular, is used for interrupting program execution (in the same way as the ABORTCF command in a command file).

Procedure nesting

Procedures can be nested, one procedure can contain a call for another one.
All the actual values to be passed to the parameters of the various procedures must be contained in the calling procedure.

The following example shows a procedure for comparing the values contained in two fields in the same collection:

```
PROC compare (a,b)
   POS FIRST ;
   REPEAT
      IF a > b THEN SHOW 'error,Ø,a,Ø,b'
      ENDIF
      POS NEXT ;
      IF $EOC THEN SHOW 'end' ;
      RETURN
      ENDIF
   END
ENDPROC
```

Another procedure is then used to open the collection and contains a call for the procedure "compare".

```
PROC openc (p1,p2,p3,p4)
   OPEN p1 MODE p2;
   compare p3 p4
ENDPROC
```

When "openc" is called the following actual values must be given in the defined order:

p1 - the name of a collection to be opened.

p2 - the opening mode.

p3 - the name of one of the fields to be compared.

p4 - the name of the other field.

For example:

```
openc NUMERIC READ FIELD_X FIELD_Y
```

The procedure "compare" can either be called by using:

```
PROC openc (p1,p2)
  OPEN p1 MODE p2;
  compare field_x field_y
ENDPROC
```

and specifying under DML:

```
openc NUMERIC READ
```

procedures and command files

A command file can be executed from a procedure and a procedure called during execution of a command file.

Execution

A procedure is to be executed under DML:
the user must specify the name it is catalogued under and the actual values for the parameters.
The actual values for the procedure must be separated from one another by blanks.
If the actual values for the procedure are expressions, they must contain no blanks; e.g. in a procedure called "evaluation", two arguments are supplied:

```
evaluation 316*5 52/7
```

Procedure actual values can be the name of a collection or of a field, a string value or more generally anything to do with an "expression" as defined in DMS (see the paragraph EXPRESSION in the chapter "DMS LANGUAGE ELEMENTS").

7. DML COMMANDS

This chapter illustrates the syntax and semantics of DML commands, indicating:

1) the command name (in alphabetic order)

2) command functions

3) command syntax

4) parameter description

5) characteristics

6) examples.

The terms defined in the chapter "DMS LANGUAGE ELEMENTS" are used in the syntactic description. The graphic conventions contained in the "Guide to the Literature" are valid.

Very often parameters associated to commands can be replaced by variables.
For example:

OPEN COLLECTION wharehouse MODE MODIFY

can be written (after allocating two variables, e.g. x and y and assigning congruent values to them):

OPEN COLLECTION x MODE y

The following list comprises the commands for carrying out these operations, and the parameters that can be replaced.

COMMAND	PARAMETERS
CLOSE	name
CONNECT	name
CREATE	name
DELETE	name
DISPLAY	identifier (with symbol ^)
EXEC	file path
EXPUNGE	name
MODIFY	name
OPEN	name; READ or MODIFY; SHARED or EXCLUSIVE
POSITION	name
PRINT	output path
REPORT	name; output path; number; string; details(s)

Expression can always be replaced by a variable.

The following examples show how useful it is to be able to generalise these problems.

The following example is of a command file:

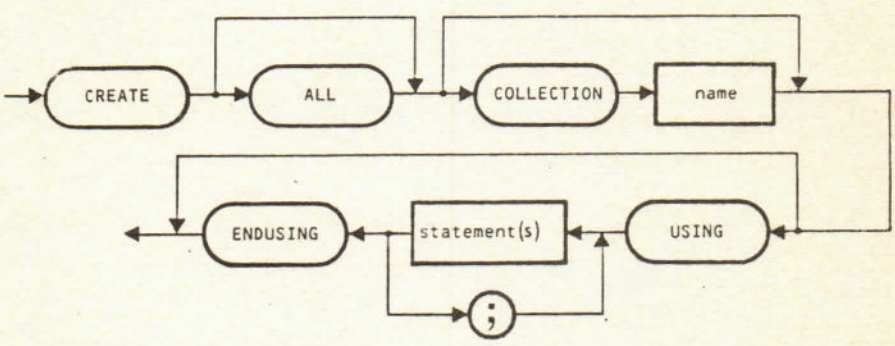
```
ALOC a,b;
MOVE $QUERY 'COLLECTION NAME:' TO %a;
MOVE $QUERY 'OPENING MODE (READ/MODIFY):' TO %b;
OPEN COLLECTION %a MODE %b;
DEALOC a,b
```

By executing this command file, the user can change the name of the collection to be opened and define the opening mode interactively.

Or, with a procedure, the user may choose to open a particular collection, and to define the parameters for preparing a REPORT on the collection in interactive mode.

```
PROC report (a,b,c)
  OPEN a MODE b MODIFIER c FILENEW;
  ALOC dev,nl,nc,t,st,exl;
  MOVE $QUERY 'DEVICE OUTPUT:' TO %dev;
  MOVE $QUERY 'LINES NUMBER:' TO %nl;
  MOVE $QUERY 'COLUMNS NUMBER:' TO %nc;
  MOVE $QUERY 'TITLE:' TO %t;
  MOVE $QUERY 'SUBTITLE:' TO %st;
  MOVE $QUERY 'expression list:' TO %exl;
  REPORT COLLECTION a TO %dev
    OPTION LINES %^nl COLUMNS %^nc
           TITLE %t SUBTITLE %st
           DETAIL %exl;
  DEALOC dev,nl,nc,t,st,exl
END
```

Displays and allows the addition of records to the current collection or to an open collection.
The command may refer to and use one or more other DML commands to specify the type of operation to be performed on the records.



where:

ALL — several records are to be created in order.
The user will be prompted for field values record by record.

COLLECTION — The open collection referred to.

name — identifies the collection.
If COLLECTION name is omitted, the current collection is assumed.

USING statement(s) ENDUSING — Allows one or more DML commands to be used for record creation.
Used for complex or repetitive operations.
statement(s) defines the DML commands that must be used.

Characteristics

The collection must have been opened in MODIFY MODE for CREATE to be performed correctly. (See OPEN command).
Records may added to positional or keyed files.

record structure

CREATE displays a record with the format described in the collection's VIEW. The following values, in order of priority serve as a guide for writing the fields:

- the initial values of the VIEW (if any)

- the values of the current record.
  Note that there is no current record only when an empty collection is opened and before creation of the first record.

- zero for numeric fields and blank for alphnaumeric fields.

  On the basis of these initial guide values, the user can assign the desired field values to the record being created.

functions keys

The user can use the following functions keys during execution of the command:

/EXIT/ adds the displayed record to the collection.
If ALL has been specified, the CREATE record sequence will continue.

/SKIP/ the displayed record is not added.
If ALL has been specified, the current record is redisplayed for creation of the next record.

/CONTROL//EXIT/ the displayed record is not added.
If ALL has been specified, the CREATE record sequence is interrupted.

The CREATE command can be executed as described above with or without the ALL clause.

If the USING clause is present, execution of the CREATE command as described above is valid only if the DISPLAY command has been specified.

USING clause

The USING statement(s) ENDUSING clause specifies commands used for record creation.
Values may be assigned directly to fields to be modified with the MOVE command or be assigned selectively and interactively by the DISPLAY command.
When the USING clause has been specified, the created records are not displayed unless DISPLAY is specified inside the USING clause.
If USING does not specify any command, a record is created with the same values as the current record.
Note that the CREATE ALL USING ENDUSING command will fill the file with records, as it cannot be interrupted.
When USING is used with ALL, the repetitive sequence can be interrupted only with the BREAK command specified inside USING.

primary key

Fields specified as primary keys in VIEW cannot be duplicated.

incorrect input

If the value entered in the field does not match the field description (in the VIEW), it is not accepted and the cursor will return to the beginning of the input field whatever function key is pressed. A correct value must be entered.

Examples

Access to a collection opened previously in MODIFY MODE but not longer current, to create a certain number of records; only certain fields are to be filled.

```
CREATE ALL COLLECTION coll2 USING
   DISPLAY surname,name,year_of_birth;
   IF surname = '*' THEN BREAK
   ENDIF
  ENDUSING;
```

The required data can be inserted in the fields indicated by DISPLAY.
Entry of '*' in the 'surname' field will interrupt the creation cycle.

When the identifier parameter is represented by a variable (see the introduction to this chapter), care must be taken to distinguish between direct and indirect access to a variable (see VARIABLE in the chapter "DMS LANGUAGE ELEMENTS").
For example, if a record has a field called field_A and the following is written:

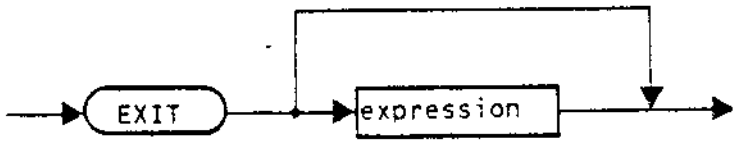
```
ALOC x;
MOVE 'field_A' TO %x;
DISPLAY %~x
```

the name of the field, and its contents will be displayed.
If, on the other hand, the following is written:

```
ALOC x;
MOVE 'field_A' TO %x;
DISPLAY %x
```

an error message is displayed.

Terminates a DMS session with re-entry to the SHELL environment. It closes all open collections.
It also allows the value specified in expression to be passed to the STATUS variable of the SHELL environment.



where:

expression — specifies a DMS expression that must be numeric (this must not be 128).

Characteristics — When EXIT is keyed-in without the expression parameter the value passed to SHELL, in the STATUS variable, is zero.

Examples

A DMS session could be closed by setting the following condition:

```
IF $ERR > Ø THEN EXIT 1
```

So if a test is carried out on the STATUS variable in SHELL environment, the value will be 1.
Or by:

```
IF $ERR > Ø THEN EXIT $ERR
```

and then by carrying out the test in SHELL environment the STATUS variable will contain the number corresponding to the error.

Permanently deletes one or more records marked for temporary deletion with DELETE or pressing the DELETE key during execution of a MODIFY command.

where:

COLLECTION — specifies the collection containing one or more temporarily deleted records to be permanently deleted.

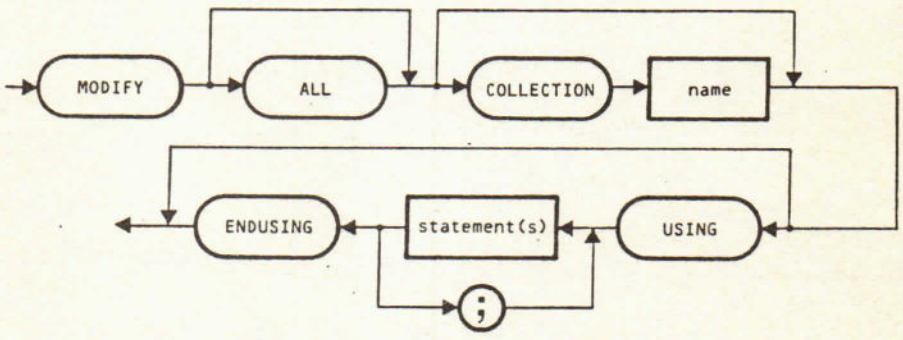
name — name of the collection.
If COLLECTION name is not specified, the current collection is assumed.

Characteristics — The collection that contains the marked records must have been opened in MODIFY MODE or EXPUNGE is not accepted.
EXPUNGE can be executed only on open record collections; the collection must not have been closed between execution of DELETE and EXPUNGE.

When the EXPUNGE command is used on files without deleted records handling (e.g. file created under BASIC) it displays:

(42) SYSTEM ERROR

Displays one or more records and allows modification or scanning of the data in the corresponding collection.
The collection may be the current collection or any open collection.
MODIFY may also refer to and use one or more other DML commands to further specify the type of modification.



where:

ALL — All the records starting from the current record are to be scanned.

COLLECTION — Specifies the open collection referred to

name — identifies the collection.
If COLLECTION name is not specified, the current collection is assumed.

USING statement(s) ENDUSING

Allows DML commands to be used for record modification especially in the case of complicated and repetitive operations.
statement(s) defines the DML commands to be used.

Characteristics

MODIFY displays the current record for modification of its fields. The collection must have been opened in MODIFY mode.(See OPEN command).

functions keys

During execution of this command,the user can control interaction with the record using the following functions keys:

/EXIT/ the modified record will be written back into the collection.

/SKIP/ the modified record is not written back into the collection.
If ALL has been specified, MODIFY prompts for the next record.

/CONTROL//EXIT/ the modified record is not written back into the collection.
If ALL is specified, scanning of the records is interrupted.

/ F3 / the current record is logically deleted and excluded from the collection while this remains open. Marked records can be permanently deleted with EXPUNGE.

MODIFY can be interpreted in this way, with or without the ALL option.
If the USING clause has been specified, MODIFY can be interpreted in this way only if DISPLAY has been specified.

USING clause

The USING statement(s) ENDUSING clause interprets the commands specified, modifying the data file directly.

When this clause has been specified, the modified records are not displayed unless DISPLAY has been specified in the clause.
Using this clause, values may be assigned directly using MOVE or interactively using DISPLAY.
If USING ENDUSING does not specify any commands, the current record is written back to the data file.
If ALL has been specified together with USING, the repetitive sequence can be interrupted only by BREAK specified within USING.

primary key

Fields specified as primary keys in VIEW cannot be modified.

incorrect input

If the value entered in the field does not match the field description (in the VIEW) it is not accepted and the cursor will return to the beginning of the input field whatever function key is pressed. A correct value must be entered.

Examples

The contents of the 'wages' field are to be updated in all the records of a collection opened previously in MODIFY but no longer current.
If the current record of the collection is not known, it is advisable to position on the first record of the collection.

```
POSITION FIRST COLLECTION coll1;
MODIFY ALL COLLECTION coll1 USING
          DISPLAY wages
          ENDUSING
```

The 'wages' field of each record is displayed with DISPLAY; functions keys are used for interactive entry.

An as-yet-unopened collection is to be accessed to modify the contents of the 'wages' field; this time the modifications are provided directly by the program.

```
OPEN coll1 MODE MODIFY;
MOD ALL USING
          IF wages < 1000
          THEN MOVE 1000 to wages
          ELSE MOVE wages + 100 TO wages
          ENDIF
        ENDUSING
```

The example below shows how the reply to a query on the video can be moved to a certain variable, using the $QUERY special symbol.

```
ALOC opt;
MOVE $QUERY 'Key in option:' TO %opt
```

'Key in option:' will be displayed and the reply to the query will be moved to the 'opt' variable.

When a DMS session has just been opened, the OPEN command must always be executed before any other commands referring to a particular collection.

During opening of a collection, a message may request the user to enter the pathname of the data file, if the data administrator has specified this in the FILE DESCRIPTOR.

When a collection has been opened in READ MODE (lower level access) and is then to be accessed in MODIFY MODE (higher level access) it must first be closed by using the CLOSE command and subsequently opened specifying MODIFY MODE. This is not necessary when going from a higher to a lower level.

The user must only specify access modes coherent with his access privileges.

current collection

During a DMS session, many different collections may be opened at the time. The last collection opened will be the current collection.
Any commands referring to open but not current collections must indicate the collection name. The name of the current collection can be displayed with the DESCRIBE command.

user messages

When an attempt it made to open a collection that is already open, 'collection already open' is displayed.
This collection now becomes the current collection and the access mode specified in the last OPEN is now valid.
When the user attempts to open a collection in MODIFY and his password allows him only READ privileges, 'collection opened in read' is displayed.
The collection is opened in this way by default.
When the user specifies FILE NEW and the data file already exists, 'file already exists' is displayed.
The collection is opened ignoring the above option.

Examples

A collection called 'coll1' is opened to create and modify records; however, the data file has only been described in the FILE DESCRIPTOR and is not yet present in the file system.
The user password must also be entered.

```
OPEN  coll1 MODE MODIFY FILE NEW PASSWORD us_a
```

The current collection is now 'coll1'.
If:

```
OPEN coll2
```

is entered, coll2 records can only be read. (Obviously, the data file referred to must already exist).

Collection 'coll2' is now the current collection.
If:

```
OPEN coll1
```

is specified subsequently, the password need not be specified as 'coll1' is already open.
As MODE has not been specified, READ is assumed, replacing MODIFY, specified previously.
Collection 'coll1' is now the current collection.

Characteristics

If the position indicated is outside the collection range, a message is displayed and the cursor is not moved. The value 1 (TRUE) is assigned to the special symbols $TOC or $EOC.

WITH clause

When the WITH clause is specified, the collection is scanned according to previously specified parameter, as follows:

- POSITION FIRST
  from the beginning to the end
- POSITION LAST
  from the end to the beginning
- POSITION PREVIOUS
  from the current record to the beginning
- POSITION NEXT
  from the current record to the end
- POSITION TO expression
  from the beginning to the end

user messages

When the position indicated is outside the range of the collection, a 'record not found' message is displayed.

Examples

An open collection containing the item 'wages' is to be scanned as follows:

```
POS FIRST WITH wages >= 1000;
IF $EOC THEN SHOW 'wages are less than 1000'
        ELSE REPEAT
             DISPLAY wages;
             POSITION NEXT WITH wages >= 1000;
             IF $EOC THEN SHOW 'end search';
             BREAK
             ENDIF
              END
 ENDIF
```

Assuming that in another simple example, we have:

```
POS TO 4Ø COLLECTION col5 WITH colour = 'dark grey'
```

The collection indicated will be scanned starting from the first record and positioning on the 40th record that satisfies the condition : colour = 'dark grey'.

In the case of a keyed file, assuming that the record has the following format:

```
      Ø1 product.
       Ø2 code            PIC X(5).
       Ø2 type            PIC X(12).
       Ø2 colour          PIC X(12).
       Ø2 price           PIC 999V99.
       Ø2 quantity        PIC 9 USAGE COMP.
```

Assuming that the 'code' field has been defined in the FILE DESCRIPTOR and VIEW descriptors as the primary key and 'price' as the secondary key:

```
POS FIRST COLLECTION woollens WITH (type = 'angora')
    AND (colour = 'dark grey')
```

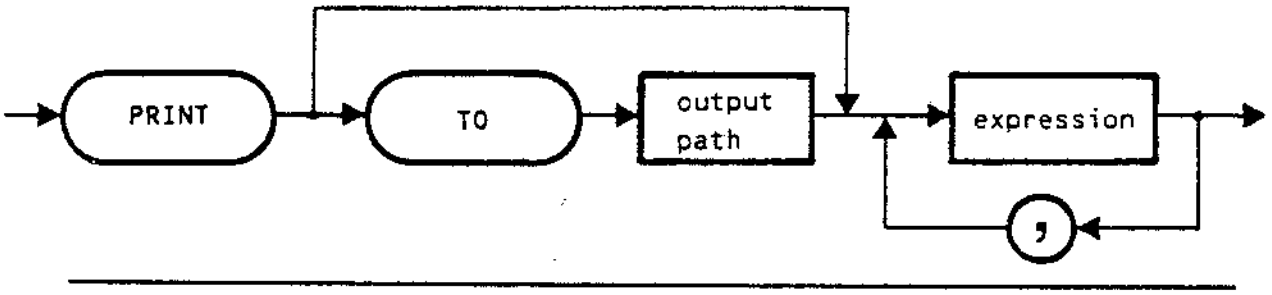
the search will be made according to primary key 'code', in ascending key value order.

Assuming also:

```
POS FIRST COLLECTION woollens KEY price
   WITH (type = 'angora') AND (colour = 'dark grey')
```

in this case, the search will be made according to secondary key 'price' in ascending key value order.

Prints the contents of 'expression' on the specified output device.



where:

TO — Sends the output data to the device indicated by output path

output path — specifies the output device which may be:

- the workstation printer, assumed by default if this parameter is not specified.
- the terminal screen indicated by the symbol $T. This device is assumed by default if this parameter is not specified and there is no workstation printer.
- the system printer indicated by the symbol $P1. If there are two system printers, the second one is indicated by the symbol $P2 according to how the system has been configured.
- a sequential file, indicated by its file system path name. If the file does not exist, it is automatically created.

- spool queues.
  This is specified by $SPOOLn, where n is a number between 1 and 4. (See the SPOOL command in the "MOS - SHELL Operating Commands - Reference Manual").

expression — Contains the data to be printed.
See the chapter on "DML LANGUAGE ELEMENTS".

Examples — With two different collections containing a common identifier, a print is required containing data from both collections.
Assuming a collection called ARTICLE with the following VIEW:

```
 Ø1 ARTICLE.
  Ø2 CODE  PIC X(5).
  Ø2 ITEM  PIC X(20).
  Ø2 STOCK PIC 9(3).
  Ø2 PRICE PIC 999V999.
```

and containing the following data:

CODE	ITEM	STOCK	PRICE
A1	T1 tractor	6	5.7
A2	T5 tractor	15	4.1
A3	P8 power cultivator	12	1.8
A4	P5 power cultivator	20	1.1

and another collection called MOVEMENTS with the following VIEW:

```
 Ø1 MOVEMENTS.
  Ø2 MVMNTS   PIC 9(4).
  Ø2 CODE     PIC X(5).
  Ø2 QUANTITY PIC 9(3).
```

and containing the following data:

MVMNTS CODE QUANTITY
1 A2 4
2 A4 5
3 A1 2

MVMNTS	CODE	QUANTITY
1	A2	4
2	A4	5
3	A1	2

and that a print with the following layout is required:

```
MVMNTS  CODE  ITEM                STOCK  PRICE(pounds)
```

when executing parallel data search in both the collections using the identifier CODE that is common to both; printing of this data can be done by using the following command file:

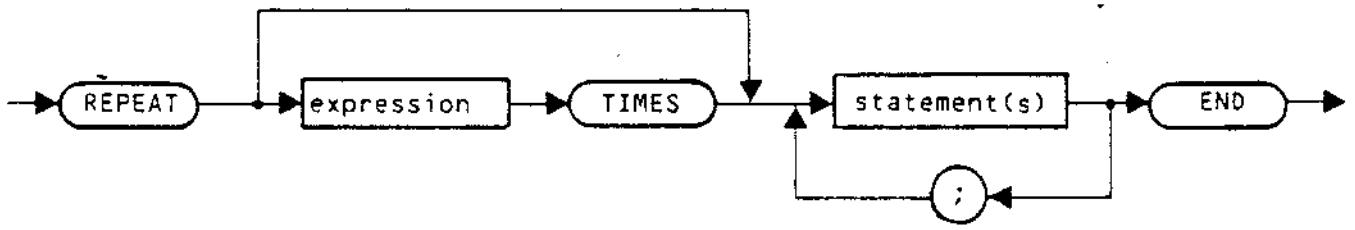
```
OPEN ARTICLE;
OPEN MOVEMENTS;
ALOC lin,head,V1,V2,V3,V4,V5;
MOVE 'ƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀƀ' TO %lin;
MOVE 'ƀƀMVMNTSƀƀCODEƀƀITEMƀƀƀƀƀƀƀƀƀƀƀƀƀƀSTOCKƀƀPRICE(pounds)' TO %head;
PRINT %lin,$CR;
PRINT %head,$CR;
PRINT %lin,$CR;
REPEAT
   POS FIRST COLLECTION ARTICLE WITH CODE=MOVEMENTS:CODE;
   IF $EOC
      THEN SHOW 'ERROR';
           ABORT
      ELSE MOVE MVMNTS $FORMAT'ƀ££££ƀƀƀƀƀ' TO %V1;
           MOVE CODE $FORMAT'£££££ƀ' TO %V2;
           MOVE ARTICLE:ITEM $FORMAT'££££££££££££££££££££' TO %V3;
           MOVE QUANTITY $FORMAT'£££ƀƀƀƀ' TO %V4;
           MOVE QUANTITY*ARTICLE:PRICE $FORMAT'ƀ£££.£££ƀƀƀƀƀ' TO %V5;
           PRINT %V1&%V2&%V3&%V4&%V5;
   ENDIF;
   POS NEXT COLLECTION MOVEMENTS;
   IF $EOC THEN DEALOC lin,head,V1,V2,V3,V4,V5;
                BREAK
           ELSE POS FIRST COLLECTION ARTICLE
   ENDIF
END
```

Execution of this file gives the following result:

MVMNTS	CODE	ITEM	STOCK	PRICE(pounds)
1	A2	T5 tractor	4	16.400
2	A4	P5 power cultivator	5	5.500
3	A1	T1 tractor	2	11.400

REPEAT

The commands specified after REPEAT are executed as a loop.
The cycle can either be repeated a specified number of times, or interrupted with a break command.



where:

expression	Specifies the number of times the REPEAT cycle of commands is to be repeated.
TIMES	is the keyword associated to 'expression'.
statement(s)	Defines one or more DML commands.

Characteristics

Repetition of the REPEAT loop can be interrupted with BREAK inside the loop itself.
REPEAT loops can be nested.
When BREAK is encountered during execution of a nested REPEAT loop, execution of this loop is interrupted. Execution continues with the loop immediately above.
A REPEAT cycle can also be interrupted with the following commands:

ABORT — to exit from execution of the commands and return to the DMS prompt.

ABORTCF — if the cycle is part of a command file

RETURN — if the cycle is part of a procedure.

Special symbol $1 is particularly useful when using the REPEAT command (see the chapter "DMS LANGUAGE ELEMENTS").

Examples

Use of BREAK and ABORT in a REPEAT loop is illustrated in the command file shown below.
In the first case, the loop is interrupted and the "end of operation" message is displayed.
In the second case, the loop is interrupted but the message is not displayed.

```
NOERROR;
POSITION FIRST COLLECTION NUME_1;
POSITION FIRST COLLECTION NUME_2;
REPEAT
    IF NUME_1:delta > 1ØØØ THEN
        MOVE NUME_1:alfa1 TO NUME_2:alfa2;
        IF $ERR THEN SHOW 'error';
                     ABORT
        END
    ENDIF;
    POSITION NEXT COLLECTION NUME_1;
    IF $EOC THEN BREAK ENDIF;
    POSITION NEXT COLLECTION NUME_2;
    IF $EOC THEN BREAK ENDIF
END;
SHOW 'end of operations'
```

The following example shows how useful the REPEAT command can be when used with visual attributes:

```
NOPROMPT;
ALOC IND,NQ,R;
SHOW $FF;
MOVE 'FORW' TO %IND;
REPEAT 20 TIMES
  IF $I>10 THE MOVE 'BACK' TO %IND ENDIF;
  IF %IND='FOR ' THEN MOVE $I TO %NQ
                 ELSE MOVE 21-$I TO %NQ
  ENDIF;
  MOVE $I TO %R;
  REPEAT %NQ*7 TIMES
     OUT $AT(%R:$I),$V(15:1)
  END
END;
SHOW 'Ø';
DEALOC IND,NQ,R
```

The following figure shows what is displayed:

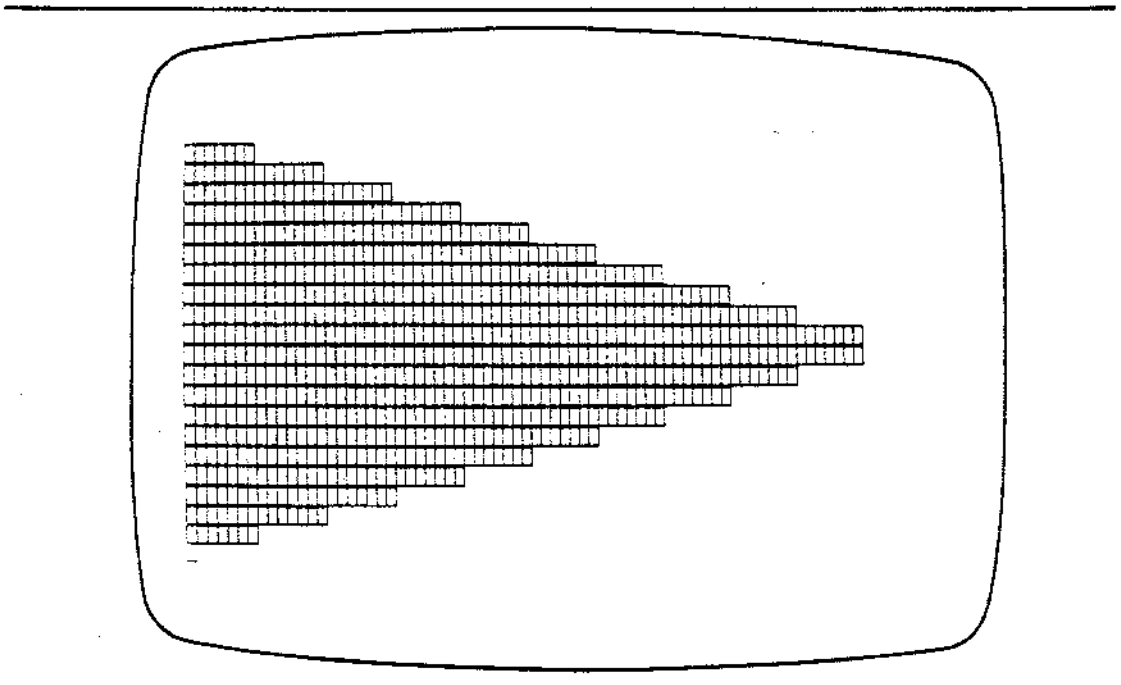
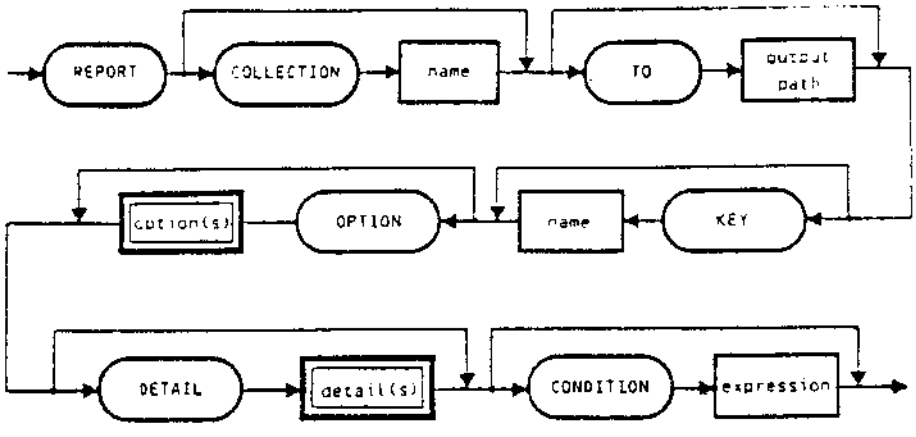


Fig. 7. 1 - The Result of Command File Execution

With this command it is possible to traverse a collection generating output for each record on a specified output device.
The report may be sorted according to the key indicated.
The output data may be printed according to page format or record format.
The REPORT command allows operations on data contained in the fields.
The user can specify criteria for accepting or rejecting certain records in the collection according to the option specified.



where:

COLLECTION — Indicates which of the open collections is referred to

name — identifies the collection.
If name is not specified, the current collection is assumed.

TO — Sends the output data to the device specified by output path.

output path — identifies the output device which may be as follows:

- the workstation printer, assumed by default if this parameter is not specified.
- the terminal screen indicated by the symbol $T. This device is assumed by default if this parameter is not specified and there is no workstation printer.
- the system printer indicated by the $P1 symbol. If there are two system printers, the second one is indicated by the symbol $P2 according to how the system has been configured.
- a sequential file, indicated by its file system path name.
  If the file does not exist, it is automatically created.
- spool queues. This is specified by $SPOOL n where n is a number from 1 to 4.
  See the SPOOL command in the "MOS - SHELL Operating Commands - Reference Manual".

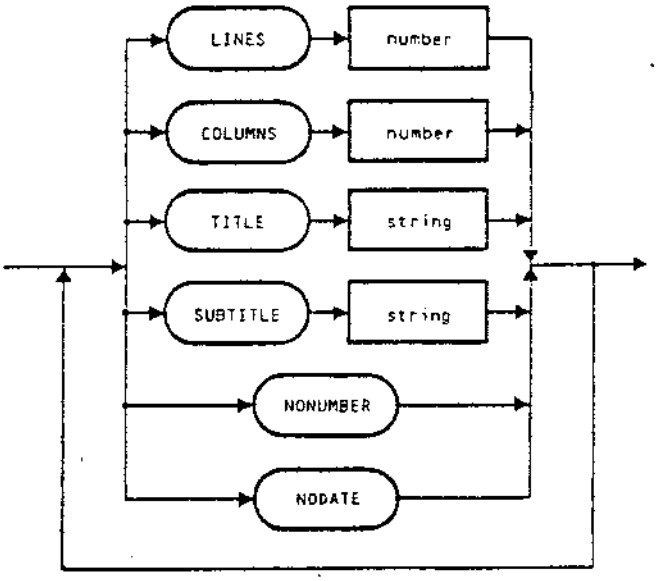
KEY — In the case of a keyed file, specifies the sort key to be used.
It may be the primary or the secondary key of the collection and must have already been described in the relative VIEW.

name — name of the field used as sort key.
If this is not specified, primary key is assumed.

OPTION	Formats the output data according to the 'option(s)' specified.
option(s)	a complete description of this parameter is given on succeeding pages.
DETAIL	Specifies record layout, selecting one or more fields and allows processing of the data they contain, according to the parameters defined in 'detail(s)'.
detail(s)	a complete description of this parameters is given on succeeding pages.
CONDITION	Evaluates each record of the collection according to the condition defined by 'expression'. If the conditions holds, i.e. it is true, the record is included in the report. If the condition does not hold, i.e. it is FALSE, the record is not included.
expression	see the chapter "DML LANGUAGE ELEMENTS".

Option(s)

May have the following meaning.
(Note that the selected options must be specified in the order as shown below from top to bottom.)



where:

LINES — Indicates the max. number of lines per output page.
number — Specifies the number of lines.

COLUMNS — Indicates the max. number of columns per output page.
number — Specifies the number of columns.
If LINES and COLUMNS are not specified, the following values are assumed:

- terminal screen 24 lines,80 columns
- printer 60 lines,132 columns
- file 60 lines,132 columns

TITLE — Page heading.
Where there are several pages, the heading is printed only on the first.

string — Contains the text of the heading (Refer to "DML LANGUAGE ELEMENTS").
If this option is not specified, the heading is omitted.

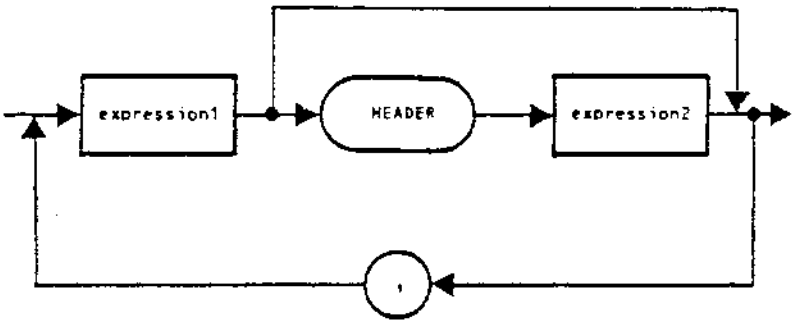
SUBTITLE — Subtitle of the output page.
Where there are several pages, the subtitle is given on each.

string — Contains the text of the subtitle (See "DML LANGUAGE ELEMENTS").
If this option is not specified, the subtitle is omitted.

NONUMBER — Numbering of the pages is not required.
If NONUMBER is not specified, pages will be numbered automatically.

NODATE — The date is not required.
If NODATE is not specified, the date and the time will be indicated on each page.

Detail(s) May have the following meanings:



where:

expression1 defines the fields and the way in which the data contained in them must be sent to output.
The text of expression1 also becomes the field header, unless this is specified differently via the HEADER option.

HEADER Modifies the field header in expression1.

expression2 contains the header of the fields referred to by expression1; evaluation of this expression must give a string result (non-numeric).

Characteristics The REPORT starts from the first record of the collection and not from the current record.

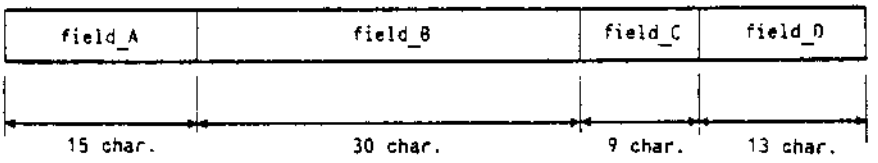
The length of the record to be printed may be longer than a print line; in this case, the output is not truncated but is transferred to the next line until all the record data have been printed.

When field headers are longer than the field length defined in the VIEW descriptor, the max. record layout value is assumed automatically.

Examples

The example below illustrates an application of the REPORT command with specific reference to the OPTION parameter.

Assuming that a record has the following elementary field structure:



When calculating the total output length, remember that REPORT will add the following blanks for each record:

- 2 blanks starting from the lefthand margin
- 2 blanks starting from the righthand margin
- 3 blanks between the single fields.

If the following value is assigned to the OPTION parameter.

```
REPORT OPTION COLUMNS 80
```

the page layout will be as follows:

FIELD_A	FIELD_B	FIELD_C	FIELD_D
item_a01	item_b01	item_c01	item_d01
item_a02	item_b02	item_c02	item_d02
item_a03	item_b03	item_c03	item_d03

If the following value is assigned:

REPORT OPTION COLUMNS 55

the page layout will be as follows:

```
FIELD_A          FIELD_B
FIELD_C     FIELD_D

item_a01         item_b01
item_c01    item_d01

item_a02         item_b02
item_c02    item_d02

item_a03         item_b03
item_c03    item_d03
```

The example below illustrates an application of the REPORT command with specific reference to the DETAIL parameter.
Assuming that reference is made to the current collection with the following fields: Surname, Name, Road, Town, Profession; if the user keys-in the following:

```
REPORT DETAIL Surname & Name $FORMAT'£££££££Ø£.'
 HEADER 'NAME',Road &'Ø-Ø'& Town HEADER 'ADDRESS',
 Profession
```

The page layout will be as follows:

NAME	ADDRESS	Profession
WELSH C.	12 Abbey Rd. - ACTON	Baker
WORSLEY P.	10 North Rd. - CHESTER	Dentist
WRIGHT H.	15 Bond St. - LONDON	Art Dealer

The example below illustrates use of the REPORT command with specific reference to the CONDITION parameter.
Assuming that reference is made to the current collection:

```
REPORT CONDITION stock < supply.
```

the report will contain all the records whose 'stock' value is lower than 'supply'.

This is only used in a procedure, and interrupts its execution.

Characteristics

This command has no options except for entry of the command itself.

Examples

Use of the REPEAT command in a simple procedure (that can be written and compiled in a DDL environment) follows:

```
PROC sum_cont (a,b)
  IF a > b THEN RETURN
  ENDIF
  SHOW a+b
ENDPROC
```

9. THE DMS.HELP FILE

The DMS.HELP file is for interactive handling of the main DML commands.
This chapter describes how this file is activated and used.

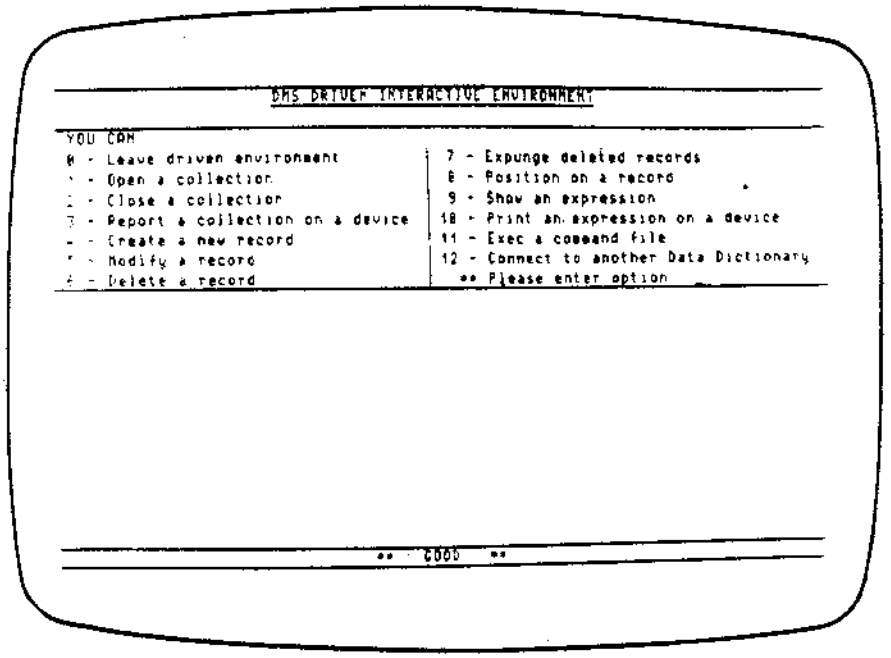
The DMS.HELP file is supplied with DMS as an optional for interactive handling of the main DML commands.

Activation

Activation of this process is carried out under DML and a REGION of at least 8000 bytes must have been specified when the session opened.
And then proceed as for a normal command file, i.e.:

```
EXEC DMS.HELP
```

The following command selection menu is then displayed:



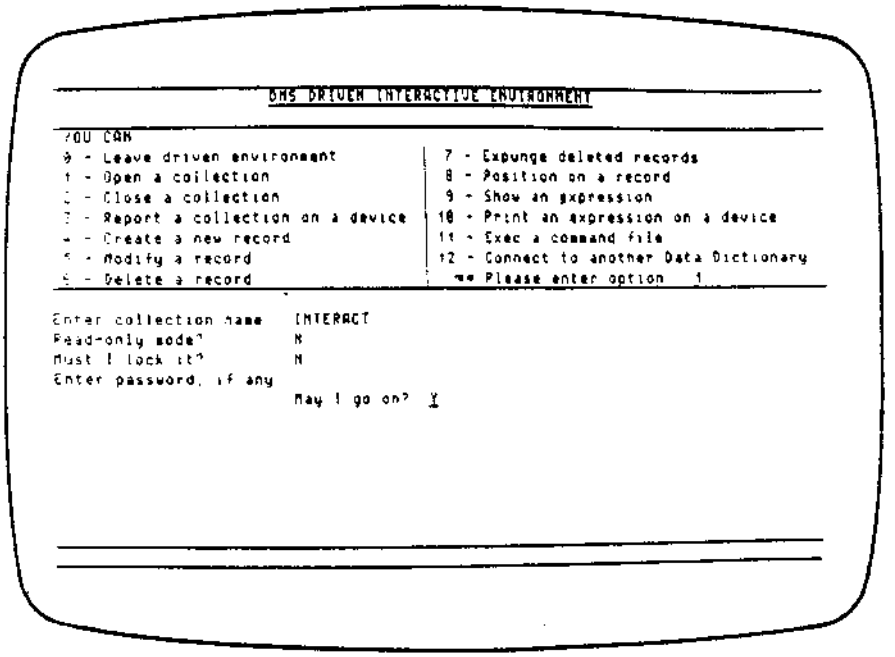
The last line is for error warnings. These are eliminated when correct input is made.

Use

When the above menu is displayed a suitable option can be entered.
If, for example, a collection is to be processed, it must first be opened.
When the option corresponding to OPEN is entered, the set of questions that follow are for definition of the OPEN command parameters.

A default value is supplied in answer to some of these questions, and these can be confirmed, if required, by keying-in the carriage return key.

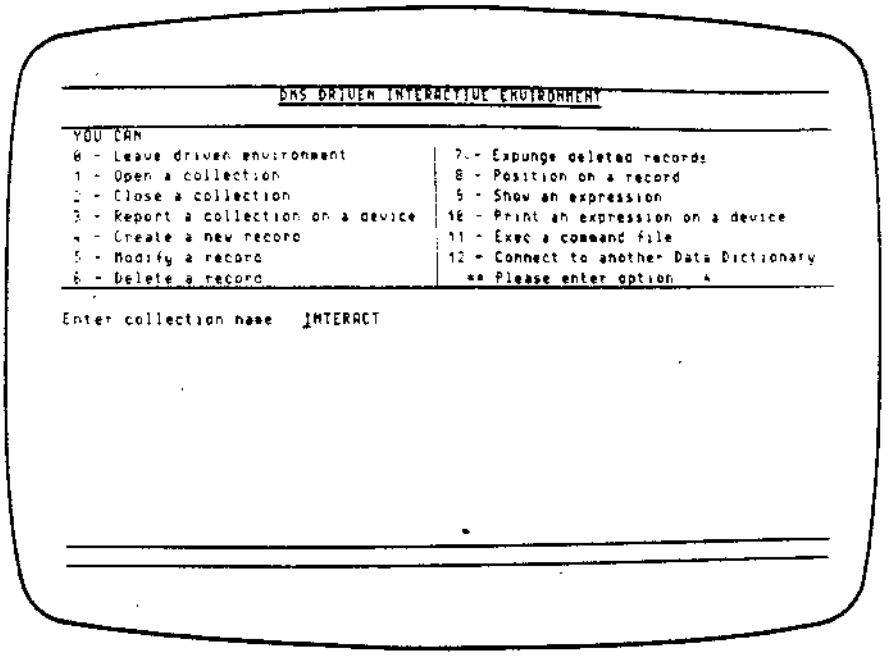
When the parameter definition phase terminates, a message is displayed to show that the system is waiting for confirmation.



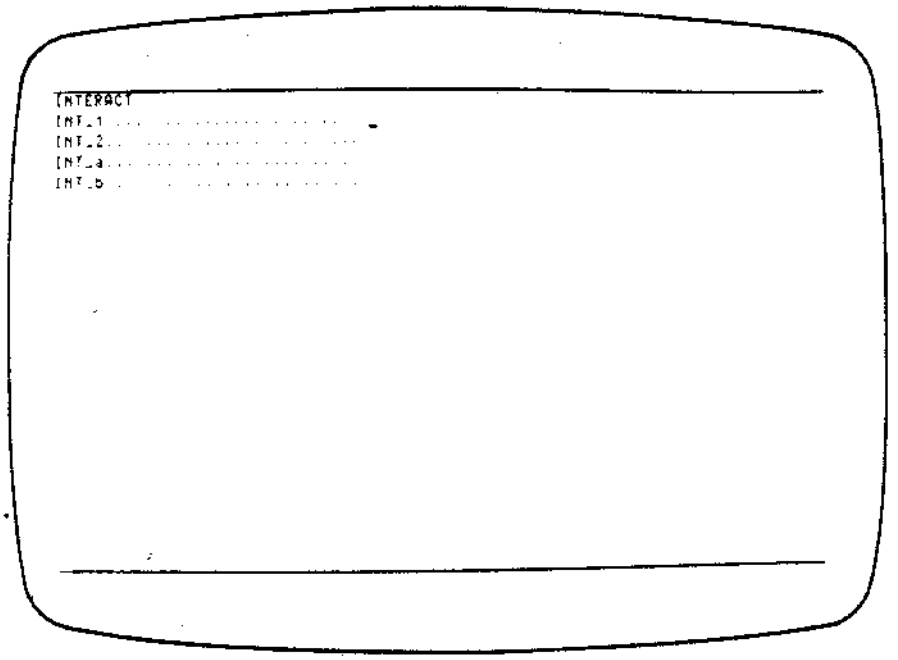
When this has been confirmed and the collection has consequently been opened in the specified way, another command can be entered.

To continue with the present example, the CREATE command is now to be used for processing.

The following request will be shown:



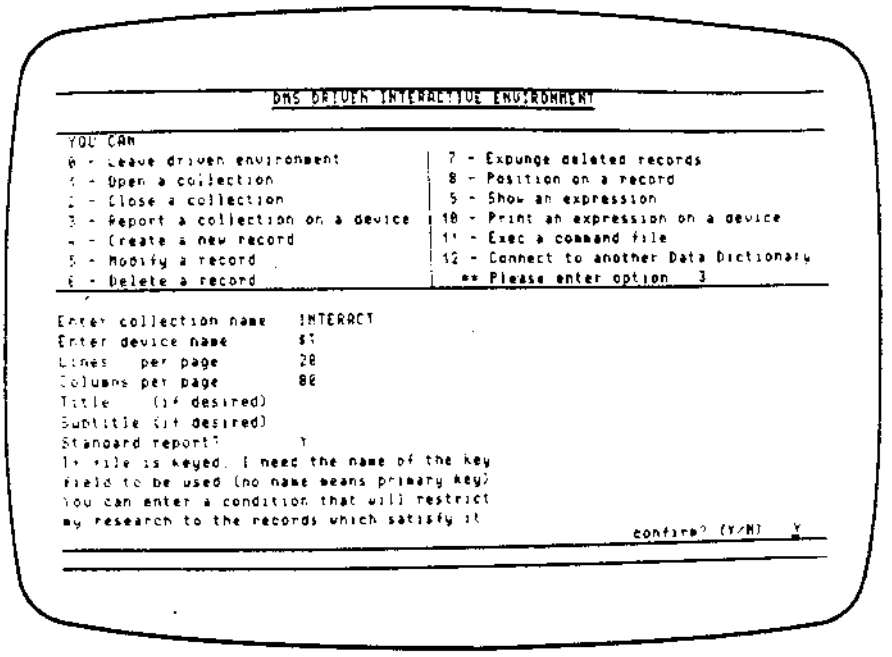
The default value assumed is that of the collection that was opened previously; if this is confirmed, on the screen the record mask is now described and the required values can now be assigned to each field, as in the following example.



So the required values can now be assigned.

The /EXIT/ key is to leave the record creation phase, and the menu is displayed again for further command selection.

If a REPORT is to be executed on the current collection, by entering the corresponding option, the following will be displayed.



The device assumed by default is the work-station printer; the same rules and symbols described for each command in the chapter on the "DML COMMANDS", are valid.
As far as the number of lines and columns per page are concerned, default values are provided that take the kind of device specification into account.

When the command definition sequence has finished and has been confirmed, the report on the data in the collection is supplied.

When the carriage return key is pressed the screen is cleared and the command menu is displayed again.

End of Operation

Option 0 must be entered to terminate the whole operation.

25	UNDEFINED VARIABLE
26	VARIABLE SHOULD BE BY REFERENCE
27	NUMERIC OVERFLOW
28	ILLEGAL FORMAT
29	VARIABLE SHOULD BE BY VALUE
30	MISSING "TO" KEYWORD
31	NO OPENED COLLECTIONS
32	DATA SET IS EMPTY
33	ILLEGAL DEVICE NAME
34	ARITHMETIC ERROR
35	ILLEGAL DESTINATION
36	RECORD NOT AVAILABLE
37	ILLEGAL VARIABLE NAME
38	NO MORE ROOM ON DEVICE
39	WRITE ERROR ON DEVICE
40	CANNOT CHANGE KEY FIELD
41	CANNOT DUPLICATE PRIMARY KEY
42	SYSTEM ERROR
43	ERROR IN ACCESSING DD
44	ILLEGAL USERID
45	ILLEGAL "USERID" KEYWORD
46	MISSING PARAMETER(S)

APPENDIX B. OPERATOR MESSAGES

This Appendix lists all the operator messages in alphabetic order.
The meaning of each message is explained and the necessary actions.

ABORT NOT ALLOWED IN 'USING' CLAUSE

ABORT cannot be used in the 'USING' option in CREATE and MODIFY commands.

ARITHMETIC ERROR An incorrect operation has been attempted.

CANNOT CHANGE KEY FIELD

An attempt has been made to modify the value of a field that has been declared a primary key. This is illegal.
The number -1 has been inserted in a field that was declared primary key. This is illegal.

CANNOT DUPLICATE PRIMARY KEY

The value contained in a field which is a primary key must be unique.

COLLECTION ALREADY OPENED

An attempt has been made to execute an OPEN command on an already open collection.

No error code is issued.

ILLEGAL FORMAT	$FORMAT has not been used correctly.
ILLEGAL NUMBER	Incorrect operand of the 'expression' element.
ILLEGAL OPTION	Syntax error on the options of the command.
ILLEGAL SYMBOL	In the 'expression' element, the $ character is followed by characters that are not 'special symbols'.
ILLEGAL USERID	The name of a user who is not enabled to access the Data Dictionary has been associated with the USERID keyword in the CONNECT command.

ILLEGAL 'USERID' KEYWORD

Incorrect USERID keyword in CONNECT command.

ILLEGAL VARIABLE NAME

An illegal 'name' has been indicated in the ALOC command for the variable to be allocated.

INCORRECT PASSWORD

The password entered does not allow access to the specified collection.

INDEX OUT OF RANGE

The index provided for a field is less than 1 or greater than OCCURS.

MISSING PARAMETER(S)

There are not enough actual values for a procedure that has been called.