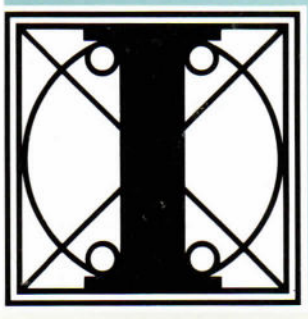
Operating System

# **MCL** MOS Command Language

## User Guide

MOS



olivetti

**PUBLICATION ISSUED BY:**

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

Operating System

# **MCL** MOS Command Language

## User Guide

olivetti

# PREFACE

This manual is a user guide to the MOS operating system Command Language (MCL). It describes the components of MCL and shows how to write MCL procedures.

This manual is intended for operators or programmers working in the Shell environment, who wish to activate programs using MCL procedures.

The manual Introduction to MOS should be read first in order to gain an understanding of the MOS operating system.

The manual SHELL Commands Reference Manual should be read in conjunction with this manual, as it describes the Shell environment and the commands which may be used within MCL procedures.

## SUMMARY

This manual has the following eight chapters:

1. Introduction
2. Lexical Structure
3. Variables and Expressions
4. Statements
5. Invoking Procedures and Programs
6. Editing MCL Procedures
7. Restart Mechanism
8. Memory Utilisation

## REFERENCES

Read first ...

Introduction to MOS (in MOS Features and Functions - Code 4041600 F)

Further information is provided in ...

DOCUMENTATION Guide - Code 4041610 U

Glossary/Glossario - Code 4002140 H

MCL MOS Command Language Pocket Reference - Code 4002670 N

MOS System Software Generation and Installation Manual - Code 4041710 W

SHELL Commands Reference Manual (in SHELL Environment User Manual - Code 4041630 W)

**FIRST EDITION:** January 1988 - Release 6.0

# CONTENTS

# 1. INTRODUCTION

MCL is short for MOS Command Language. It provides an easy-to-use interface to the MOS operating system, which interprets it by means of the Shell component.

MCL allows the construction and execution of simple Pascal-like procedures. It can handle both system and user-created programs, submitting them as jobs to the operating system.

This manual describes the basic features of MCL. These are as follows:

- a compact and adaptable lexical structure (see Chapter 2)
- the capacity to handle and evaluate variables and expressions (see Chapter 3)
- the ability to accept high-level assignment, control and built-in statements and so to allow the construction of MCL procedures (see Chapter 4)
- the capacity to nest and subsequently invoke commands, procedures and programs, with the passing of either positional or keyed parameters (see Chapter 5)
- the ability to run syntactic checks on new MCL procedures and report any errors found (see Chapter 6)

# 2. LEXICAL STRUCTURE

This chapter describes the characters and combinations of characters available with MCL.

## LANGUAGE CHARACTERS

MCL contains the following alphanumeric characters:

- the letters A through Z
- the numbers 0 through 9.

In addition it accepts as alphanumeric the following characters:

/ the slash
- the underline character
. the period.

This allows you to include any of these characters in a file path name without having to put the path name in string quotes. (See Introduction to MOS for more information on file path names.)

As well as this standard alphanumeric set, MCL can accommodate a number of national character sets. The particular set that a system requires is defined in the Shell configuration file (SYS/$CONVSH), which can be modified by the user.

The following characters are also accepted by MCL:

| | | | | | | |
|---|---|---|---|---|---|---|
| + | - | * | = | > | < | ? |
| ( | ) | ; | & | : | ' | |
| " | % | , | ^ | $ | # | |

The characters ",", "^", "$" and "#" must be enclosed by quote (') marks.

The characters "$" and "#" may have alternative characters substituted for them in certain national character sets, as shown in the following table:

| ASCII VALUE | | NATIONAL EQUIVALENT | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DECIMAL | HEXADECIMAL | USA | ITALY | FRANCE | GREAT BRITAIN | GERMANY (ORIGINAL) | GERMANY (WEST) | SPAIN | PORTUGAL | DENMARK | SWEDEN FINLAND | NORWAY | SWITZERLAND FRENCH | SWITZERLAND GERMAN | GREECE | YUGOSLAVIA |
| 35 | 23 | # | £ | £ | £ | # | # | £ | # | £ | # | £ | £ | £ | £ | # |
| 36 | 24 | $ | $ | $ | $ | $ | $ | $ | $ | $ | ¤ | $ | $ | $ | $ | ¤ |

## SPECIAL SYMBOLS

A number of characters and combinations of characters have a special significance to MCL. These are the symbols that establish the beginning and end of alphanumeric strings and their relationship to other strings.

The special symbols allowed in MCL statements are as follows:

```
:=  assignment symbol
;   statement separator
>   output director (overwrite)
>>  output director (append)
<   input director
<<  include-file identifier
%   variable prefix
"   optional parameter prefix
```

The special symbols allowed in MCL expressions are as follows:

```
=   equals sign
<>  not equals sign
>   greater than sign
<   less than sign
>=  greater than or equals sign
<=  less than or equals sign
+   add sign
-   subtract sign
*   multiply sign
&   concatenate sign
'   string constant delimiter
(   start of expression sign
)   end of expression sign
(*  start of comment sign
*)  end of comment sign
?   identifier for any character sequence
```

## KEYWORDS

Certain alphabetic character strings have a fixed meaning in MCL. These keywords are as follows:

| | | | |
|---|---|---|---|
| AND | ELSE | IF | RESUME |
| BEGIN | END | NOT | SET |
| BM | ENDRST | ONRST | THEN |
| DIV | EXIT | OR | TO |
| DO | FOR | RETURN | WHILE |

The keywords may be modified or substituted with alternative keywords.

The first letter of a keyword can be in position 1 of a line. The last letter of a keyword can be in the last position of a line. Keywords in other positions must be preceded and followed by one blank position.

## STRING CONSTANTS

MCL accepts as a string constant any sequence of zero or more characters.

If the string constant contains a special symbol, or consists of zero characters (an empty string) it must be enclosed in single quotes.

An arithmetic operator can use a string constant containing only numbers.

## IDENTIFIERS

An MCL identifier can be any alphanumeric string of 1 through 159 characters.

The following are all examples of valid identifiers:

a/b/c

12

a.b.c

## INCLUDE FILE IDENTIFIERS

Include file identifiers are particular types of identifiers. They are made up of alphanumeric strings prefixed by the characters <<.

When the Shell encounters the MCL characters << it interprets the string following them as a file path name. It then replaces the complete include file identifier with whatever is contained in the file. These files must be byte-stream. It also adds a space before and after the included file text.

For example, if you enter the following:

```
%a12   <<a/b/c   %c+1
```

and the contents of the file "a/b/c" is the text "%c2+%d", the Shell interprets your entry as the following:

```
%a12   %c2+%d   %c+1
```

This type of identifier cannot be nested.

## SPACES

Blank spaces are freely allowed in all MCL procedures. However, you must not type spaces in the middle of:

- variables
- identifiers
- numeric constants
- keywords.

## COMMENTS

Comments can be inserted anywhere in an MCL procedure.

Anything enclosed in "(*" and "*)", even if it takes up more than one line on the screen, is recognised to be a comment, e.g. (*TEST PROGRAM*).

It is not possible with EDPARSE to modify an instruction containing a comment longer than one line.

## EXPANSION OF A FILE NAME

The character "?" matches any string of characters (except "/"), including an empty string.

When Shell encounters the character "?" in a file identifier or path name, it either searches in the working directory, or starts from the root (if the specified path name starts with the character "/") and selects all the files whose name is compatible with the string.

The character "?" is usually employed for passing positional parameters and calling programs or procedures.

Examples of reference to the working directory:

| | |
|---|---|
| ? | matches all its filenames |
| ?obj? | matches all its filenames which contain "obj" |
| ?mp | matches all its filenames which end in "mp" |
| A? | matches all its filenames which start with "A" |

**Note:** The character "?" can be specified one or more times, in one section only of a complete path name.

Example:

/AAA/B?C?A/D : correct

/A?/BB?/GGG : incorrect

# 3. VARIABLES AND EXPRESSIONS

This chapter describes how to establish variables and expressions with MCL.

## VARIABLES

An MCL variable identifier is any sequence of 1 to 160 alphanumeric characters immediately preceded by a percentage sign (%), carriage return inclusive.

There are no declaration statements for MCL variables. Variables are created locally in the procedures in which they occur, and have no existence outside them.

MCL variables can only assume string values. The Shell initialises each variable with the empty string at the start of a procedure. Variables then assume whatever string values the procedure assigns to them.

A variable can be used as the operand of an arithmetic operator, but only if its current value is a numeric string.

You should not attempt to invoke within a called procedure any variable that belongs to the calling procedure. Nor should you attempt to return any values from a called to a calling procedure, other than by means of the RETURN statement (see Chapter 4), or the predefined variable %RET.

Chapter 4 describes how to use assignment statements to establish values for variables. Chapter 5 describes how to use positional and keyed parameters to pass values to variables.

## PREDEFINED VARIABLES

In addition to the local variables that you create yourself there are a number of predefined system variables. These are as follows:

| | |
|---|---|
| %OPT | Contains the string passed as an option when the procedure was activated. |
| %PAR | Contains the number of positional parameters in the current MCL procedure invocation. |
| %PATH | Contains the list of directories in which the commands issued are searched. |
| %PROMPT | Contains the current MCL prompt. |

| | |
|---|---|
| %RET | Allows a string value to be passed from a called procedure or program to the calling procedure. |
| %STATUS | Allows a completion code to be passed from a called procedure or program to the calling procedure (see Chapter 5). |

You can assign values to these predefined variables as you can to user-created variables. %PAR and %STATUS, however, can assume values without an explicit assignment, and %PATH and %PROMPT have default values reassumed at each new login.

**%PATH Variable**

The %PATH variable identifies the succession of directories the Shell searches when you type a file name without including its complete directory path.

Using the SET assignment statement you can include any valid directory path name in the %PATH variable. Write complete path names in the order in which they are to be searched, separating each with a blank space. Place the complete string within quote marks. In the %PATH variable, there may be a maximum of 16 directory pathnames and each individual pathname may have a maximum of 59 characters. For example:

```
SET %PATH := '/IPL/CMD . /IPL/BIN'
```

The default value for %PATH is as follows:

```
/IPL/DPC/CMD /IPL/DPC .
```

where CMD represents the command directory, DPC is the dependent components directory and the character '.' represents the working directory. This means that the Shell searches for a file first in the command directory, then in the dependent components directory and only looks in your working directory if this search proves fruitless.

The working directory is set for each user at login time, as specified in the login database (see the command MKLOGIN in MOS System Software Generation and Installation Manual). If this file does not exist, the working directory is set to the local root "/" by default. Example:

```
/IPL/USR/user
```

where "user" represents the name of the user.

To alter the working directory use the built-in statement SETWDIR.

### The %PROMPT Variable

By default, the Shell provides you with the prompt "MCL:". Each time it is ready to accept fresh input from the terminal keyboard it displays this prompt.

The %PROMPT variable allows you to modify the current Shell prompt. To do so, type a SET assignment statement when the existing prompt is displayed on the screen.

Example:

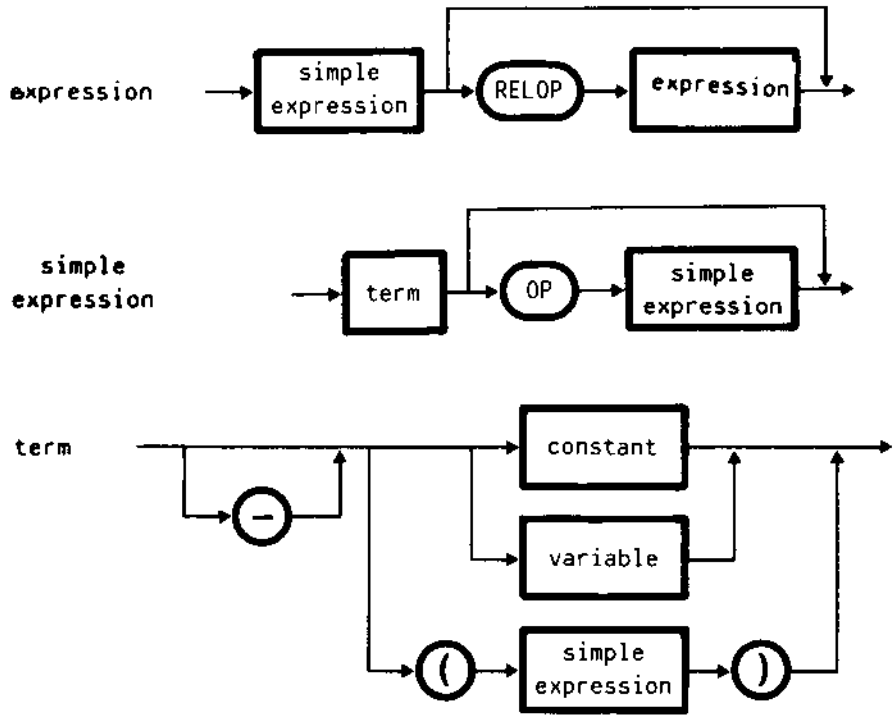
```
SET %PROMPT := READY
```

This statement converts the Shell prompt immediately to "READY". It will remain like this until a fresh assignment statement modifies it, or until logout.

After a logout the Shell prompt returns to the default. To make a prompt reassignment permanent you must include it in a .INIT file (see SHELL Commands Reference Manual for information on a .INIT file).

## EXPRESSIONS

An MCL expression can be defined by the following syntax:




where:

**RELOP** represents a relational operator

**OP** represents a dyadic operator.

### Relational Operators

The relational operators available with MCL are as follows:

| | |
|---|---|
| = | equals |
| <> | does not equal |
| > | is greater than |
| < | is less than |
| >= | is greater than or equal to |
| <= | is less than or equal to |

### Dyadic Operators

The dyadic operators available with MCL are, in order of priority, the following:

| | | |
|---|---|---|
| * | multiplied by | numeric operators |
| DIV | divided by | |
| + | plus | |
| - | minus | |
| & | concatenated with | alphanumeric operator |

### Simple Expressions

Simple expressions possess values as long as the correct dyadic operators are used with their operands. The only operator that functions with non-numeric operands is the concatenation symbol (&). All others produce run-time errors when they encounter non-numeric characters.

Run-time errors, which are not indicated on the screen, also occur when the value of a numeric variable exceeds 12 digits, or when the result of an operation is a number outside the range $-2^{31}$ to $2^{31}-1$.

### Parentheses

Where a simple expression contains more than one dyadic operator, you may use parentheses to indicate the order in which the Shell is to perform its operations. A term placed within parentheses is always processed first.

If brackets are not specified, operations are executed according to their priority.

Example of a simple expression:

```
%a * (%b + %c)
```

Example of valid expressions and equivalent to each other:

```
%a * %b + %c

(%a * %b) + %c
```

**Boolean Expressions**

Expressions occurring within the syntax of IF and WHILE statements (see Chapter 4) return a boolean value.

With an expression that contains a relational operator, the value TRUE is returned when the indicated relationship proves correct. When it proves incorrect the value FALSE is returned.

The only relational operators that are valid with non-numeric strings are "=" and "<>". With numeric strings all the relational operators are valid.

Where an expression is a variable, the value FALSE is returned when the variable contains the empty string. Otherwise the value TRUE is returned.

Boolean values can only be arrived at by evaluating boolean expressions. They cannot be assigned to MCL variables.

**Logical Operators**

The following operators are available with MCL:

NOT
AND
OR

The logical operators may only be used with boolean expressions. An expression containing a logical operator returns an empty string if the result of the operation is FALSE, or a non-empty string if the result is TRUE (usually a blank character). This value may be assigned to any MCL variable.

# 4. STATEMENTS

This chapter describes the ways of writing statements that are understood by the Shell. The following types of statements are described:

- assignment statements
- control statements
- built-in statements.

They are placed in alphabetical order for ease of reference. See the DOCUMENTATION Guide (Chapter 3) for an explanation of the syntax diagram format.

The semicolon that concludes each syntax diagram is the separator that is mandatory for dividing one statement from the next. After the last statement in a statement sequence you should not type a semicolon but press **CR**.

## ASSIGNMENT STATEMENTS

Assignment statements are constructed using the MCL keyword SET.

## CONTROL STATEMENTS

Control statements are constructed using the following combinations of MCL keywords:

```
BEGIN ... END
EXIT
FOR ... TO ... DO
IF ... THEN
IF ... THEN ... ELSE
ONRST ... ENDRST
RETURN
WHILE ... DO
```

## BUILT-IN STATEMENTS

MCL built-in statements provide a range of functions for manipulating strings and simple expressions. They are executed directly by the Shell.

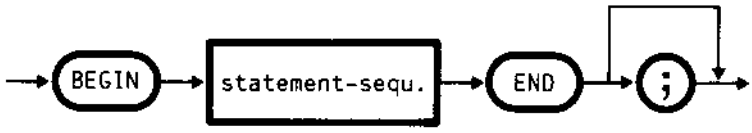
The MCL built-in statements are as follows:

| | |
|---|---|
| BM | RDPOS |
| CONN | READ |
| CONSP | RIGHTS |
| DISCON | SETWDIR |
| ECHO | SHCON |
| EDPARSE | SHNAME |
| LASTCOM | SHVAR |
| LENGTH | SHWDIR |
| LOGOUT | SUBSTR |
| PRIOR | TRACE |
| RDNEXT | WRITE |

In addition to these statements each user program in executable format and all the Shell commands can be included in an MCL user procedure (see SHELL Commands Reference Manual).

Reserved Shell commands can only be included in a procedure activated by the system administrator (user name ROOT).

The BEGIN and END keywords allow you to group a sequence of MCL statements into a single syntactic statement.




where:

**statement-sequ.** is a series of MCL statements separated by semicolons.

### Characteristic

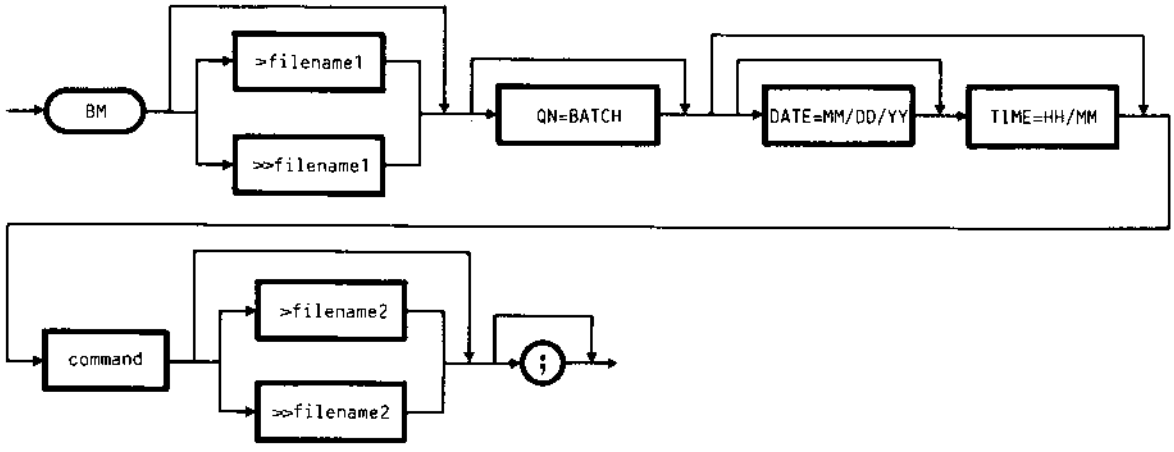
You may but need not put a semicolon between the last MCL statement and the END keyword.

### Example

```
IF %a THEN SET %a := 1
      ELSE BEGIN
             SET %b := 2;
             SET %c := 3
           END
```

Submits the specified job to the batch environment. The command will then be executed in background under the control of the batch system (see BATCH in SHELL Commands Reference Manual ).

A unique number is returned by the batch system to identify the submitted job. This number is displayed on the video.




where:

**command** is any command string which may be executed under the Shell environment: MCL procedure and executable program

**filename1** is the file to which the output of the command is redirected (path name = 59 characters maximum); the type of redirection is specified by:

> to replace the contents of "filename1"

>> to append the output to "filename1"

**BATCH** specifies, in a distributed configuration, that the job must be submitted to the global batch queue; if none is present, the job is submitted to the local batch queue

**MM/DD/YY** represents the date, in the format month ("MM"), day ("DD"), year ("YY"), in which the submitted job should be performed; if this is omitted, the current day is the default

**HH/MM** represents the time, in the format hours ("HH") and minutes ("MM"), at which the submitted job should be performed

**filename2** (see filename1 and explanation below).

**Redirection of Output**

Output can be redirected in four ways.

1. BM >filename1 command

   The specified file will contain output from both the batch Shell environment and from the executed command.

2. BM >filename1 command >filename2

   This differentiates between output from the batch Shell environment, which is redirected onto filename1, and that from the command, which is redirected onto filename2.

3. BM command >filename2

   If only filename2 is specified, output from the command will only be redirected onto this file. Output from the batch Shell environment will be redirected onto a temporary file.

4. BM command

   If no redirection mode is specified the output being displayed is lost. Output to the system line (via WRITE) or to a file, is carried out as usual.

Files specified in the command BM must be identified by their complete path names (59 characters maximum). In a cluster configuration, these path names identify files on the machine from which the batch command is issued, or on another machine if that machine name is given in the path names. For example,

```
BM >/IPL/OUT SHDIR /DEV >>/IPL/OUT1
```

identifies local output files, and

```
BM >/../NOM2/IPL/OUT SHDIR /../NOM2/DEV >>/../NOM2/IPL/OUT1
```

identifies output files on machine 2 of network 0, which may be a remote machine, or the same machine from which the batch command is issued, in which case this example is identical to the one above.

In a local area network, the **BATCH** option may be used to specify that the job should be submitted to the global batch queue rather than the local batch queue.

The working directory and the variable %PATH for the batch environment may be specified in the command line to be executed.

**Characteristics**

1. BM procedures cannot be nested, i.e. the command line executed in batch mode must not contain the keyword BM.

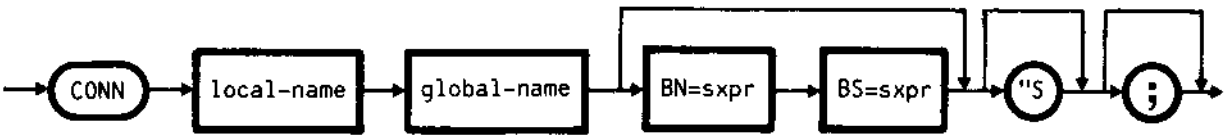
2. The directory in which a batch job is executed is the working directory from which the job was submitted to the batch environment.

3. The jobs which are submitted with an indication of the date and time for their execution are placed in a specific queue for all the jobs which are date and time dependent.

Connects a local name used in a program with a global name recognised by the file system. The local name of the program will be associated with the global name by the run-time support of the language. In this way, a general program can be written whose connections are defined before execution.

Local name and global name are connected until either disconnection using the DISCON command, or a logout.

A special type of global name is that predefined in SPOOL1 - SPOOL16, which correspond to the spooler classes respectively. These names allow direct appending of user program output files to the spooler classes. When a local name and one of these predefined names are connected, a temporary file is associated to the latter. When the program terminates, this temporary file is appended to the respective spooler class, and the identifying number of the job to be printed is returned. It is then possible to activate the procedure again, creating a new temporary file associated to the local name for the output file.




where:

**local-name** is the local name in the program to be executed; it must have no more than 14 characters and must not contain "/"

**global-name** is the name or the path name of the physical entity of the file system; it must have no more than 60 characters

**BN=sxpr** indicates the number of buffers needed in the private buffer pool; the maximum number is defined in the configuration file

**BS=sxpr** indicates the dimension of each private buffer in bytes

**"S** is the option to request synchronous I/O.

**Characteristics**

1. I/O operations on a file are usually carried out using a system buffer pool. The parameters of the CONN command (BN=sxpr and BS=sxpr) are used to define a private buffer pool to assign to the global object in the command (global-name). These parameters can only be used once for each file system object and can never be used to move buffers from one private buffer pool to another. If another user has already opened a file on which to carry out I/O operations, the request for a private buffer pool is not accepted.

2. The system carries out all I/O operations on files in asynchronous mode. This means that requests for I/O are accepted by the system immediately and the user process does not have to wait for the I/O operation to finish, before regaining control. If the option "S is specified the user program in execution after an I/O request only regains control when the I/O operations terminate. If data is appended to a file, changing its dimensions, the file is updated with the new dimensions in synchronous mode.

3. The maximum number of connections to the same or to different spooler classes is 3.

**Note**

See also the commands:

- CONSP
- DISCON
- SHCON

**Examples**

1. The executable program RECEIPTS (written in PASCAL, or BASIC, etc.) contains two local names: FILE1 and FILE2. Before execution of the program, these local names must be connected to existing file system objects:

```
CONN FILE1 /IPL/USR/FRANK/ACCOUNTS/CLIENTS;
CONN FILE2 /IPL/USR/FRANK/ACCOUNTS/ORDERS;
RECEIPTS;
  .
  .
  .
DISCON FILE1;
DISCON FILE2;
```

2. If the user wants to define his own buffer pool and carry out synchronous I/O, he can specify:

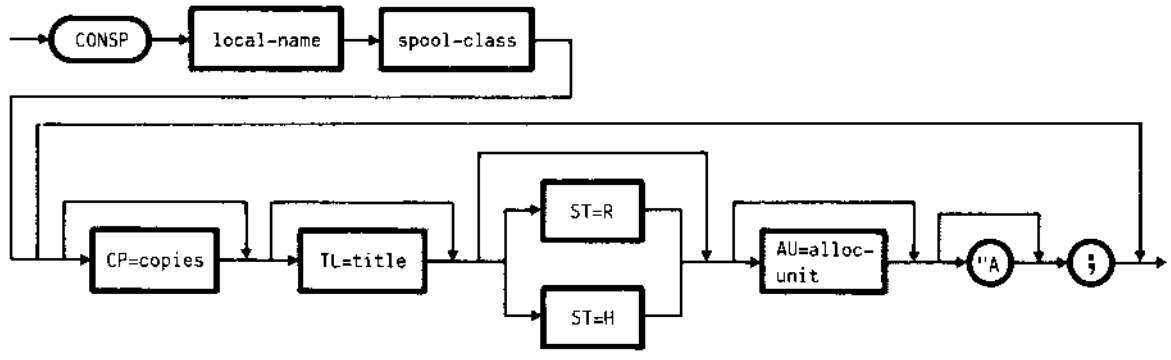
```
CONN FILE1 /IPL/USR/FRANK/ACCOUNTS/CLIENTS BN=3 BS=1024 "S
```

Connects a local name used in a program to a SPOOL class.

This allows the output file from a user program to be appended directly to a spooler class.

When a local name is connected to a spooler class, a temporary file is associated to the latter. When the program that uses the local name as output file terminates, this temporary file is appended to the appropriate spooler class, and the identifying number of the job to be printed is returned. It is then possible to activate the procedure again, creating a new temporary file associated to the local name for the output file.

The local name is connected to the spooler class until the connection is released using the DISCON command.




where:

**local-name** is the local name in the program to be executed; it must have no more than 14 characters and must not contain "/"

**spool-class** is the SPOOL class to which the print file is appended; the values accepted are in the range SPOOL1 - SPOOL16, or SP1NxMy - SP16NxMy where "x" is the network number and "y" the machine number

**copies** gives the number of copies to print; if this value is omitted, one copy only is printed by default (maximum of 99)

**title** is the title of the print job; its maximum length is 40 characters, and any blanks included in the title must be enclosed in quotes; if this is omitted, the local name is given as title to the print job

**alloc-unit** is the allocation unit of the file temporarily created by the spooling mechanism; if it is omitted, 8192 is assumed by default

ST gives the print job status:

R = READY

H = HOLD

If this is not specified READY is assumed by default

"A is the option to request paper alignment on the printer.

**Characteristics**

1. If option "A is specified, a message requesting paper alignment is sent to the master workstation (global or local), or to the workstation to which the printer is connected, when the file is about to be printed.

2. Three connections at most can be made to the same or different spooler classes.

**Note**

See also the description of the following commands:

- DISCON

- SHCON

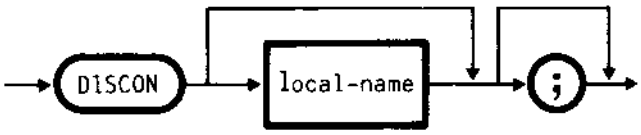
**Example**

The executable program INVOICE (written in PASCAL, or BASIC, etc.) contains two local names: OUTPUT1 and OUTPUT2, which are the output files of the program itself. Before the program is run, these local names must be connected to a SPOOL class, so that they can be printed subsequently.

```
CONSP OUTPUT1 SPOOL3 CP=2 TL='PRINT INVOICE' "A;
CONSP OUTPUT2 SPOOL3 TL='MONTHLY INVOICE';
INVOICE;
  .
  .
  .
DISCON OUTPUT1;
DISCON OUTPUT2;
```

Disconnects one previously connected local name.

If no parameters are specified, all the local names will be disconnected.




where:

**local-name** is the local name used in a previous connection.

**Note**

See also the commands:

- CONN
- CONSP
- SHCON

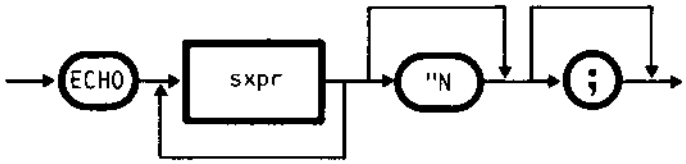
## ECHO

Displays the strings resulting from the calculation of simple expressions which are specified in the command at execution time.

The keyed parameters are ignored. The automatic line feed may be disabled during the execution of this command.

This command cannot be redirected.

---




---

where:

**sxpr** is a simple expression, the result of which is the string displayed

**"N** is the option for disabling the automatic line feed.

### Characteristic

This command can be used both interactively and in MCL procedures.

Used within an MCL procedure, it allows the operator to follow the execution of the procedure itself. ECHO commands can be inserted in several points in a procedure whereupon a series of messages will appear on its execution.

Used interactively, the command can be used for calculation. The result of the requested expression will be displayed on the line immediately following the command.

### Note

See also the commands:

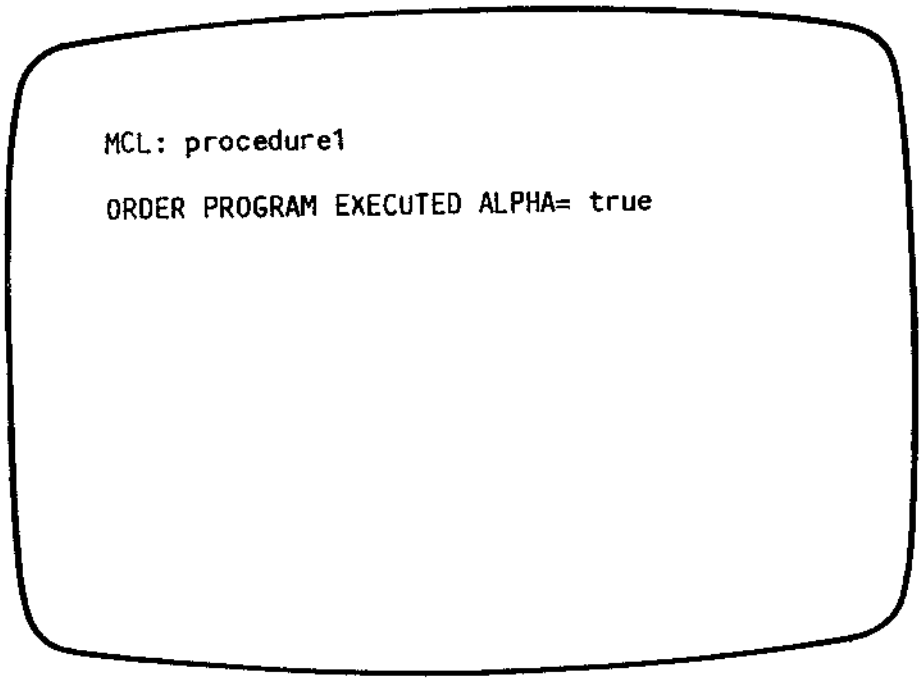
- SHOW (see SHELL Commands Reference Manual)
- WRITE

**Example**

An MCL procedure called "procedure1" contains the following command:

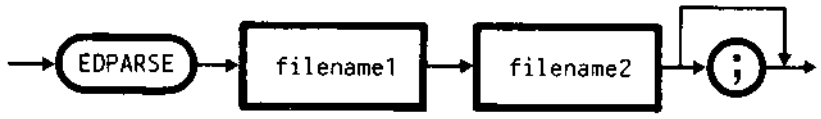
```
ECHO 'ORDER PROGRAM EXECUTED ALPHA='&%ALPHA ;
```

where %ALPHA is a variable defined in the procedure itself. When the procedure is activated and the command is executed, the resulting display is as follows:



## EDPARSE

The EDPARSE statement initiates a syntactic check of a specified procedure.




where:

**filename1** represents the name of the procedure you want to check

**filename2** represents the destination of the checked procedure.

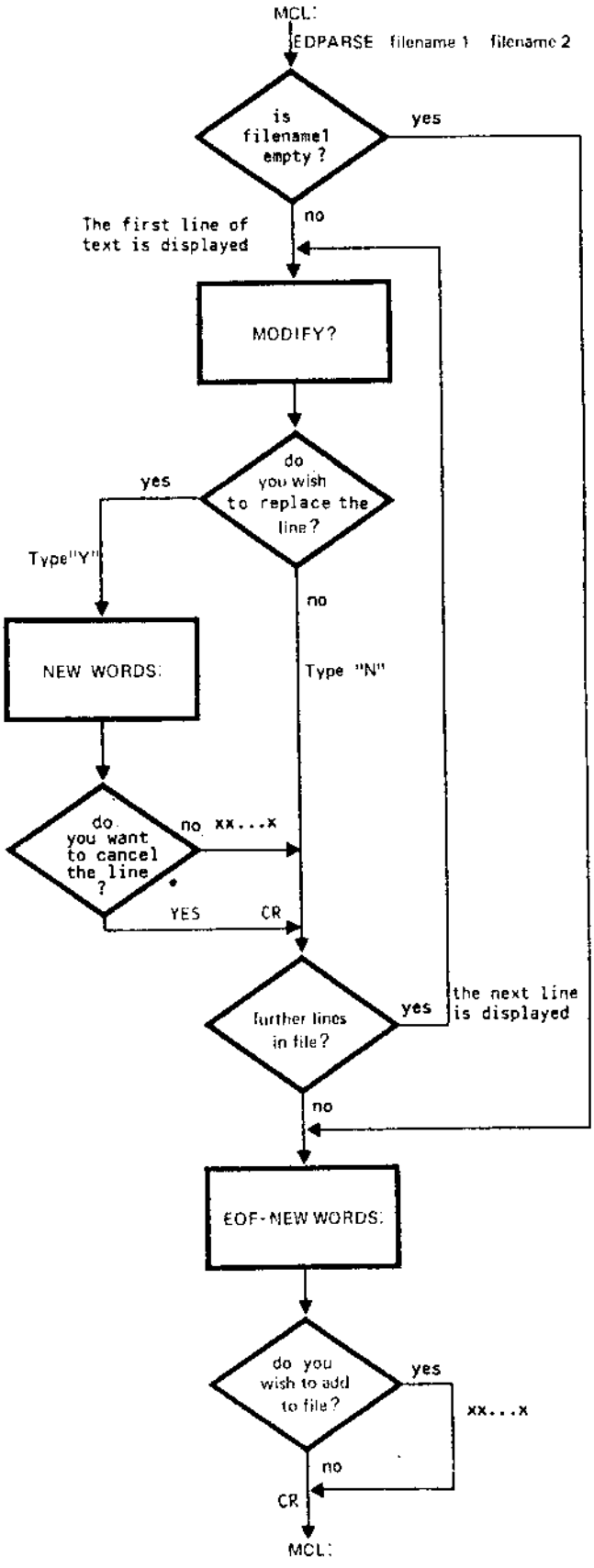
### Characteristics

1. EDPARSE displays one line at a time and prompts for any modifications you may wish to make. Accept or edit these lines as you wish.

2. Before displaying the next line, EDPARSE checks the syntax of the modified or unchanged line. If a syntax error is found, an error message is displayed and EDPARSE exits to Shell.

3. After you have used EDPARSE you are left with two files. The original file (filename1) remains unmodified, while the destination file (filename2) contains the newly edited text. This appears in full if no errors were found, truncated at the point of the error if one has been found.

4. In order to analyze a procedure containing an include file identifier, the identifier must be hidden in a comment. This can be done by using the character pairs "(*" and "*)" before and after the identifier.


MCL:
EDPARSE filename 1 filename 2
is filename1 empty ?
yes
no
The first line of text is displayed
MODIFY?
do you wish to replace the line ?
yes
Type"Y"
no
Type "N"
NEW WORDS:
do you want to cancel the line ?
no xx...x
YES CR
further lines in file ?
yes
the next line is displayed
no
EOF-NEW WORDS:
do you wish to add to file ?
yes
xx...x
no
CR
MCL:

MODIFY?

This message asks you if you want to alter the line of text that appears on the screen.

| | |
|---|---|
| N | Leaves the line as it is. |
| Y | Deletes the line and prompts you to enter a new line of text. |

NEW WORDS:

This message indicates that you may enter a new line of text which will replace the deleted line.

| | |
|---|---|
| xx...x | Enter the new line. You may type up to 159 characters followed by a carriage return. |
| CR | The deleted line of text is not replaced with new text. |

EOF - NEW WORDS:

This message indicates that you may enter a new line of text which will be appended to the file.

| | |
|---|---|
| xx...x | Enter the new line. You may type up to 159 characters followed by a carriage return. |
| CR | No new line is added to the file. |

The EXIT keyword allows the termination of a WHILE/DO or FOR/TO/DO control statement, before the normal condition for termination is encountered.

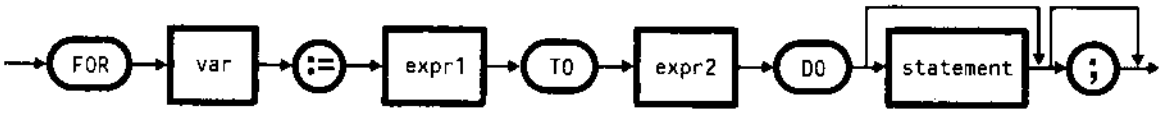

The EXIT keyword can be used to retain control of a procedure when an error condition is detected. For example, in the control statement:

```
WHILE %A DO
   IF %ERROR THEN EXIT
             ELSE PROC ;
```

the execution of the procedure PROC requires that the variable A% is a non-empty string and that the variable %ERROR is an empty string. The variables %A and %ERROR may be set within the body of PROC.

## FOR/TO/DO

The FOR/TO/DO keywords allow the repetitive execution of a statement. This execution is made dependant on the value of a variable which is incremented from an original value to a final value. One execution of the statement to be repeated is performed for each value of the incremented variable.




where:

**var** is any numeric variable

**expr1** is any valid expression resulting in a numeric value

**expr2** is any valid expression resulting in a numeric value

**statement** is an MCL statement which will be executed a number of times equal to the value of expr2 - expr1 + 1; omission of the parameter only slows execution of the procedure.

### Characteristics

1. If expr1 is greater than expr2, the statement will never be executed.

2. If expr1 and expr2 are equal, the statement will be executed once.

### Example

```
FOR %A := 1 TO 10 DO ECHO %A ;
```

The IF/THEN/ELSE keywords allow the conditional execution of one statement, or the choice between execution of two statements.


where:

**expression** is a boolean expression returning a value of TRUE or FALSE (see Chapter 3 for the rules governing the construction of expressions)

**statement1** is an MCL statement executed if the value of the boolean expression is TRUE

**statement2** is an MCL statement executed if the value of the boolean expression is FALSE.

**Characteristics**

1. To avoid ambiguity in nested IF statements, each ELSE is regarded as dependent upon the last IF before it. For example:

   ```
   IF E1 THEN IF E2 THEN S1 ELSE S2
   ```

   is understood by MCL to be equivalent to:

   ```
   IF E1 THEN BEGIN
              IF E2 THEN S1 ELSE S2
              END
   ```

   To make the second statement dependent on the first expression you would have to write:

   ```
   IF E1 THEN BEGIN
              IF E2 THEN S1
              END
         ELSE S2
   ```

2. When using the ELSE condition make sure you do not insert a semicolon before the ELSE keyword. This would terminate the IF statement prematurely.

3. The expression to be evaluated may contain repetition of one logical operator. For example:

   ```
   IF A AND B OR C
   ```

   is not permitted, but the following are allowed:

   ```
   IF A OR B OR C

   IF A AND B AND C
   ```

**Examples**

1. ```
   IF %b < 10 THEN SET %a := 1
              ELSE SET %a := 0;
   ```

2. ```
   IF %b THEN SET %a := %b + %a;
   ```

This command allows the user to modify the size of the History List (HL), or to display it.

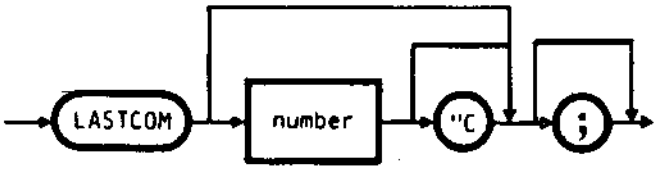
## THE HISTORY LIST

The history list retains a number of previous commands, as specified by the user using LASTCOM.

Commands need not have been executed, and may be invalid, but all are included in the history list in the order they were entered.

Each command is automatically given a sequence number starting from 1, also retained in the history list.

A command still on the history list may be re-executed by quoting its sequence number, or by quoting a string contained in it, and may also be modified before execution (see the sections on Command Repetition and Command Editing in the SHELL Commands Reference Manual ).




where:

**number** specifies how many commands are to be retained in the history list (maximum of 200)

**"C** causes the next history list sequence number to be shown alongside the prompt.

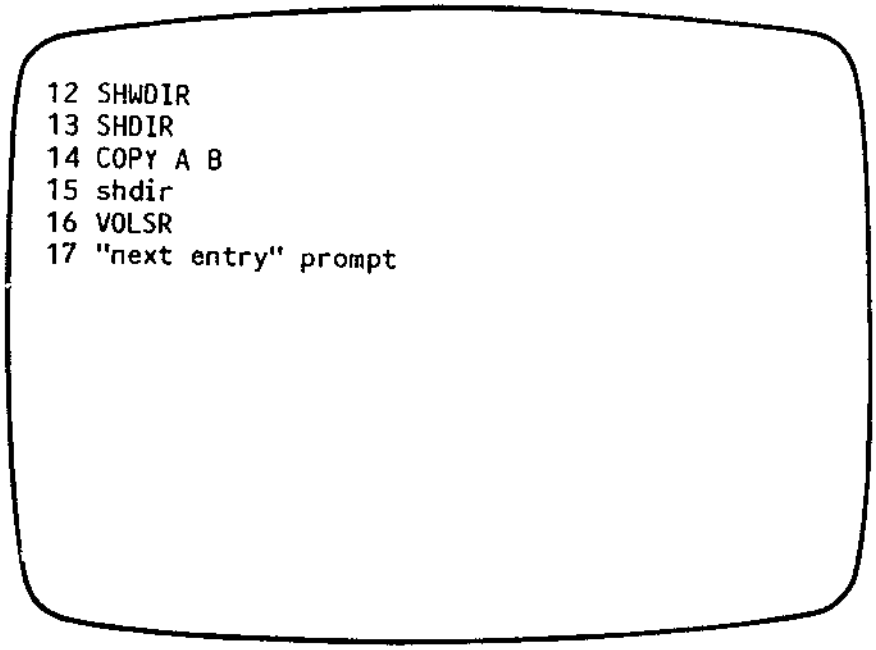
If no parameters are entered, LASTCOM displays a list of previous commands and their reference numbers, which are automatically allocated (see history list example below).

**Example**

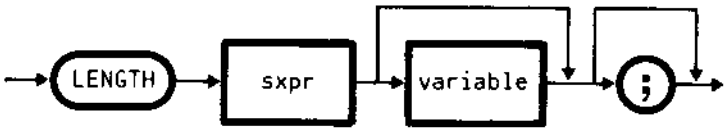
LASTCOM 100

Limits the History List size to the 100 latest commands.

**History List Example**



Calculates the length of a string resulting from evaluating a given simple expression. The length is assigned to the variable specified in the command line, if any. If no variable is given, the result is displayed on the video.




where:

**sxpr** is a simple MCL expression evaluated to give the string whose length is required

**variable** is an MCL variable to which the numerical value of the string length is assigned.

**Example**

```
SET %C:='ALFA';
SET %A:=12;
SET %B:=34;
LENGTH %C&(%A+%B) %LENGTH ;
SHVAR %LENGTH ;
6
```

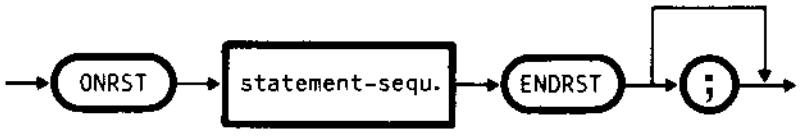
%A+%B is 46, therefore %C&(%A+%B) is ALFA46 which has a length of 6 characters.

## ONRST/ENDRST

The keywords ONRST/ENDRST are used to group a block of instructions into a single syntactic instruction, which is executed during restart. It is ignored during normal execution of the MCL procedure.

This instruction is used to handle anomalous interruption of a procedure (system crash) during restart.

---




---

where:

**statement-sequ.** is a sequence of MCL instructions separated by ";".

### Characteristics

1. Any number of ONRST...ENDRST instructions can be inserted in an MCL procedure.

2. Use of keywords ONRST and ENDRST is only allowed at first level nesting of a procedure. If this is attempted at another level an error message is emitted.

3. This instruction may not be activated interactively, but only from within an MCL procedure.

4. It is not possible to activate an MCL procedure from the system line with a restart request.

5. Keywords ONRST and ENDRST cannot be used in a BEGIN...END instruction: otherwise, execution would resume from the keyword ONRST when restarting, and an error message would be emitted as a result of the unexpected END keyword.

### Note

See Chapter 7 for details on the restart mechanism.

**Example**

In the following example the application programs or procedures executed may be of any type.

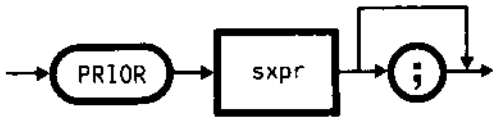
```
ONRST
  ECHO ' PROGRAM1 RESTART ';
ENDRST;
PROGRAM1;
ONRST
  ECHO ' PROGRAM2 AND PROGRAM3 RESTART '
ENDRST;
PROGRAM2;
PROGRAM3;
ONRST
  ECHO ' PROGRAM4 RESTART '
ENDRST;
PROGRAM4;
```

If procedure execution continues correctly the instructions between ONRST and ENDRST are ignored.

If there is a system crash during execution of PROGRAM3, for example, Shell restarts the program and executes the instruction ECHO ' PROGRAM2 AND PROGRAM3 RESTART ' and execution restarts from PROGRAM2.

## PRIOR

Decreases the priority of all subsequently started programs, until PRIOR is executed again.

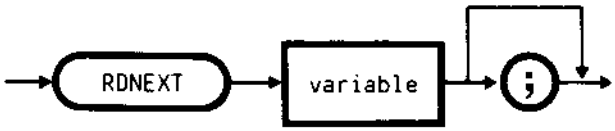



where:

**sxpr** is a simple expression which defines the value by which the priorities of all subsequent programs are to be decreased (a higher value means a lower priority).

The value of sxpr must be >= zero. If sxpr is zero, the priority is reset to the standard value. A negative value is ignored.

The RDNEXT statement assigns to a specified variable the next positional parameter in a parameter list.




where:

**variable** obeys the rules for the construction of variables described in Chapter 3.

**Note**

See Chapter 5 for more information on parameters.

**Example**

After the activation of the procedure:

```
PROCEDURE 200 1000
```

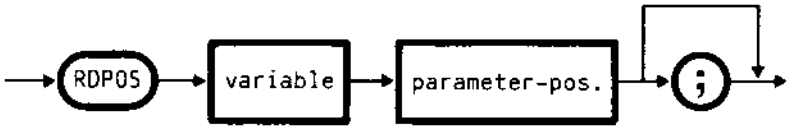
where the procedure contains:

```
RDNEXT %number;
RDNEXT %price;
```

the variable %number will contain 200, and the variable %price will contain 1000.

## RDPOS

The RDPOS statement assigns to a specified variable the value of a positional parameter selected by number.




where:

**variable** is to contain the value of the positional parameter

**parameter-pos.** represents the numeric position of the selected parameter.

### Characteristic

If the position 0 is specified, the complete path name of the procedure in execution is assigned to the variable.

### Note

See Chapter 5 for more information on parameters.

### Example

After the activation of the procedure:

```
PROCEDURE 12 512 1000
```

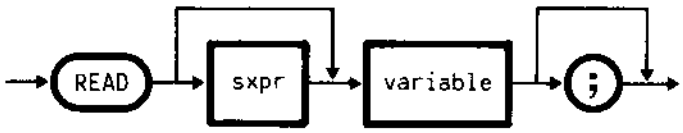
where the procedure contains:

```
RDPOS %b 1;
RDPOS %a 3;
```

the variable %b will contain 12, and the variable %a will contain 1000.

This function reads an input line (<=160 characters including carriage return) and stores this in a specified MCL variable.

If the two parameters are specified, the first is issued as prompt and the second is the MCL variable in which the input line is stored.




where:

**sxpr** is a simple MCL expression which returns a string that is displayed

**variable** is the name of the MCL variable in which the input line is stored.

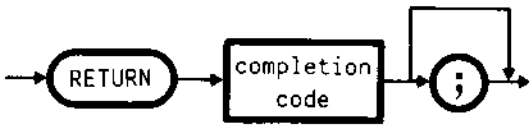
**Example**

The MCL procedure called KEY is being defined:

```
CLEAR;
READ 'FILENAME = ' %A;
READ 'RECORD LENGTH = ' %B;
READ 'KEY START = ' %C;
READ 'KEY LENGTH = ' %D;
READ 'USER CODE = ' %E;
READ 'ALLOC UNIT = ' %F;
READ 'SPLIT (50,90,100) = ' %G;
READ 'PAGE SIZE (512,1024,2048,4096) = ' %H;
MKKEYED %A RCDLG=%B START=%C KEYLG=%D USR=%E AU=%F SPLIT=%G PSIZE=%H;
```

# RETURN

The RETURN keyword allows the termination of the current procedure and a return of control to the calling procedure.




where:

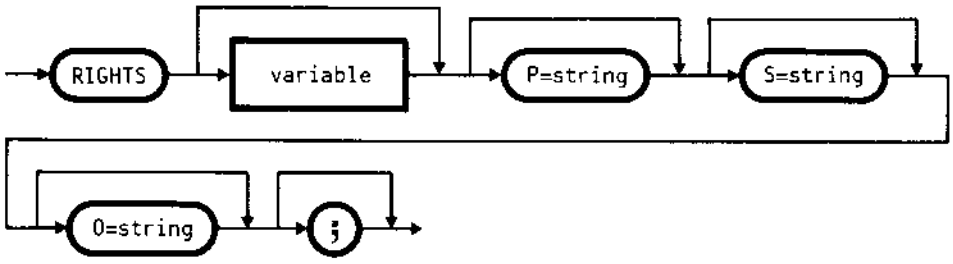
**completion code** is a value returned to the calling procedure; it obeys the rules for the construction of simple expressions described in Chapter 4.

## Characteristics

1. The RETURN statement, when activated, assigns the value of the specified simple expression to the predefined system variable %STATUS. This value can then be manipulated within the calling procedure.

2. All Shell commands return a completion code in the variable %STATUS to indicate the outcome of their operation. See under "Completion Codes" in Chapter 5.

Enables the user to read or modify the default access rights assigned by the file system when a file is created.




where:

**variable** is the name of the MCL variable in which the access rights are read. If no variable name is specified, the access rights are displayed on the screen

**P=string** is the string containing the owner's access rights (see characteristics)

**S=string** is the string containing the access rights of the group members (see characteristics)

**O=string** is the string containing the access rights of all other system users (see characteristics)

**Characteristics**

If the default access rights are to be modified, one of the following values can be specified in the "string" for the input parameters **P** , **S** , and **O** :

- any combination of the characters **R** for read access right, **A** for append access right, **W** for write access right, and **X** for execute access right

- the character _ (underline), which cancels all access rights.

**Examples**

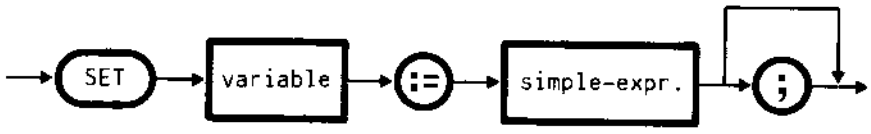
1. RIGHTS ;

2. RIGHTS %A ;

3. RIGHTS P=RW O=R ;

# SET

The SET keyword lets the user write an assignment statement, giving a value to a user-created or predefined variable.




where:

**variable** is a string preceded by the % symbol, obeying the rules for the construction of variables described in Chapter 3

**simple-expr.** is a simple expression as defined in Chapter 3.

## Characteristic

There may be blank spaces between the elements of an assignment statement, as in the first of the examples below. However, remember not to type spaces within any of the elements of an assignment statement.

## Examples

1. SET %A := %A + 1;

2. SET %PROMPT:= '#';

3. SET %MESSAGE := 'FIRST STEP EXECUTED';

4. A string defining a command may be assigned to a variable. For example:

   SET %SWD := SHWDIR;

   The command will then be activated simply by entering %SWD.

5. Commands which require parameters may be redefined in the same way. For example:

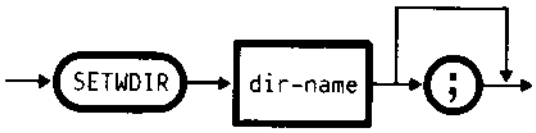
   SET %SH := SHDIR;

   allows the command to be activated by entering:

   %SH dir-name

   where dir-name is the path name of the directory to be displayed.

This replaces the current working directory with that specified in the command.




where:

**dir-name** is the name or full path name of the new working directory.

### Characteristics

To be able to access the directory specified by "dir-name", the user must have execute access right on it.

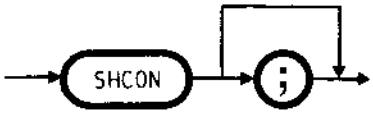
### Examples

1. SETWDIR /USR/TIM/AMM/DIP;

2. SET %AMM:=/USR/TIM/AMM;
   .
   .
   .
   SETWDIR %AMM;

## SHCON

Displays all the connections of the local or remote names.




The command has no parameters.

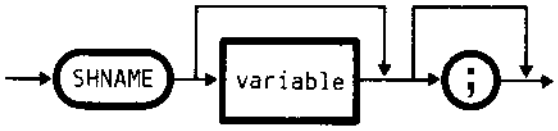
**Example**

```
MCL: SHCON

FILE1   /IPL/USR/MARILYN/RAF
FILE2   /DD/J
FILE3   /ABC/LAR

MCL:
```

This function stores the login name of the user connected to the terminal from which the command is activated, in the MCL variable given. If no variable is specified the login name of the user is displayed.




where:

**variable** is the name of the MCL variable to contain the login name.

**Example**

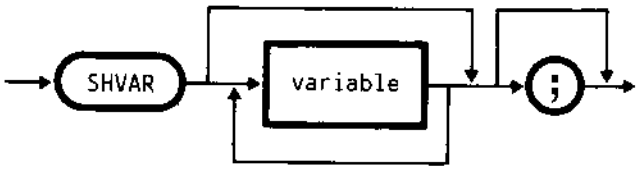
```
      .
      .
      .
MNT FL2 /DUMPVOLUME ;
SHNAME %NAME ;
COPY '/USER/'&%NAME /DUMPVOLUME "R ;
      .
      .
      .
```

## SHVAR

Displays the value of the specified MCL variables. These may be predefined variables or ones created by the user.

If no parameter is specified, all the existing variables which have a significant value will be displayed.

---




---

where:

**variable** is the name of the MCL variable to be displayed.

**Example**

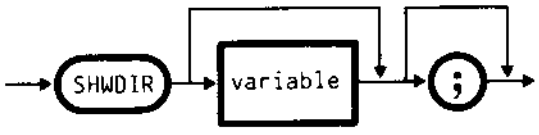
```
MCL: SHVAR

     $stdin  TTYA
     $stdout TTYA
     PATH  /CMD .
     PROMPT  MCL:

MCL:
```

Displays the path name of the current working directory.

If a variable is specified, the path name of the current working directory is stored in this variable.




where:

**variable** is the name of the MCL variable.

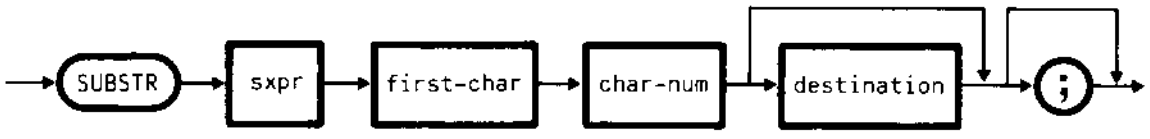
## SUBSTR

Displays the substring extracted from a specified string. This string is the result of a simple expression specified in the command.

The user must specify the displacement of the first character of the substring with respect to the start of the string, and also its length.

If the "destination" variable is specified in the command, the substring is stored in that variable.

---




---

where:

**sxpr** is a simple expression which results in the string from which the substring is extracted

**first-char** is the position of the first character of the substring with respect to the start of the string; it can be a simple expression that returns the numeric value

**char-num** is the number of characters of the substring; this can be a simple expression that returns the numeric value

**destination** is a variable which will contain the substring.

**Example**

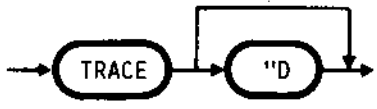
```
SET %A := 'NCOFRA';
SET %B := 'NKBUS';
SUBSTR %A&%B 4 5 %NAME;
SHVAR %NAME;
FRANK
```

Constructs a trace file in which to read and write temporary information in synchronous mode.

This file is created in the home directory which is active straight after login, with the name "IT$username" if the user is working interactively, or "BT$numId" if working in batch mode ("numId" being the name of the job).

Shell also loads the TRACE identifier that is attached to the file "IT$username" or "BT$numId", in the program context table.

---




---

where:

**"D** is the option for removing the trace file and for deleting the name "TRACE" from the context table.

**Characteristics**

1. The trace file is always removed automatically when the user exits from Shell using the LOGOUT command.

2. A trace file is searched for automatically at login. If one exists it means that there has been a system crash; in fact the user has exited from environment without using the LOGOUT command. If this is so, Shell inserts the identifier TRACE, which was deleted from the context table during the crash.

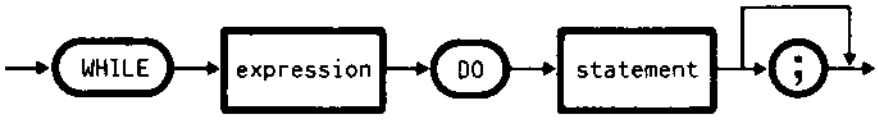
**Example**

```
TRACE;
FOR %I := 1 TO 10 DO
  BEGIN
    SHOW %I >>TRACE;
     .
     .
     .
  END;
TRACE "D;
```

If a system crash occurs, the operator can use MORE to read the TRACE file, and from its contents find out where to restart the interrupted procedure.

## WHILE/DO

The WHILE/DO keywords allow the repetitive execution of a statement. This execution is made dependent on the satisfaction of a condition.




where:

**expression** is a boolean expression returning a value of TRUE or FALSE (see Chapter 3 for the rules governing the construction of expressions)

**statement** is an MCL statement executed repetitively if and as long as the value of the boolean expression is TRUE.

### Characteristic

If the value of the boolean expression is initially FALSE, the statement is not executed at all.

### Example

```
WHILE (%a + 1) < 10 DO SET %a := %a + %b;
```

Writes on the system line the strings resulting from the calculation of simple expressions or keyed parameters specified on the command line, when this is executed. Only the value of keyed parameters is displayed.

The number of characters resulting from the evaluation of the expression must not exceed 160.

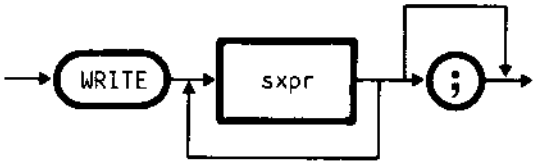
If the total sum of characters including carriage return is greater than 78, the strings are stored in the log file .MESG and the following message appears on the system line:

THERE IS A MESSAGE

The .MESG file may be displayed using the LIST or MORE command.

The command may not be redirected.

---




---

where:

**sxpr** is a simple expression, the result of which is the string displayed.

**Characteristics**

1. This command may be used within an MCL procedure as well as interactively as long as the login mechanism for accessing the system is present.

2. When used within an MCL procedure, this command enables the user to follow the procedure execution. Indeed, by inserting WRITE statements at various points of the MCL procedure, the user may follow its execution from the display of the different messages on the system line.

3. Interactively, this command is used in the same way as a calculator. Having entered the expression to be calculated the result is displayed on the system line.

**Note**

See also the commands:

- ECHO
- SHOW

**Examples**

1. WRITE 'phase 100 executed';
2. WRITE 100 * 5248;
3. WRITE %A DIV 3418;

# 5. INVOKING PROCEDURES AND PROGRAMS

This chapter describes the available ways of executing combinations of the MCL statements described in Chapter 4. It also describes how to run executable programs by means of the Shell.

In addition to the statements described in Chapter 4, the Shell can accept a range of operating commands. These can be invoked directly from the keyboard or included in procedures. See SHELL Commands Reference Manual for full details of these commands.

## DIRECT MCL INPUT

You can execute MCL statements directly from a terminal keyboard.

An MCL input line is a list of MCL statements separated by semicolons. It can contain up to 159 characters and is concluded with a carriage return.

You can type as many input lines at a time as you wish. Indicate the end of your input by typing the LOGOUT command to leave the Shell environment.

## MCL PROCEDURES

An MCL procedure is a byte-stream file containing MCL input lines.

You may type as many input lines into a file as you wish.

To invoke a repeatable procedure, type the name of the file that contains it. Follow the file name with whatever parameters you need to make the procedure specific to your needs (see the next section for more information on parameters).

See SHELL Commands Reference Manual for details as to the repetition of the activation command or for any modification.

MCL files can also be invoked from within direct MCL input and from within other invoked files. Files can be invoked both by their actual names and by simple expressions which, when evaluated, give the actual names.

You can specify the full directory path of an MCL file starting with the root. Alternatively you can specify any subtree that connects this file with one of the directories specified in the predefined variable %PATH. In this case do not precede the directory path with a '/'. For example, if the working directory is as follows:

/IPL/USR/TIM

and the complete path name of a file is:

/IPL/USR/TIM/DOCS/SHELL

then, providing the working directory is listed in %PATH, the file can be invoked simply by typing:

DOCS/SHELL

You may wish to change the working directory (particularly if you wish to invoke a number of procedures that share the same directory path). To do this use the SETWDIR command.

A special form of parameter allows input/output redirection. The standard input and output of MCL procedures are always the keyboard and screen. However, you can redirect input or output for procedures or programs nested within an MCL procedure, by supplying the appropriate parameter to the calling procedure. The input/output redirection mechanisms are described later in this chapter.

## RUNNING PROGRAMS

The Shell provides the interface that allows you to invoke executable programs. Like MCL statements, these can be invoked interactively or from within MCL procedures.

To invoke a program, type the name of the file that contains it. Either specify the complete path name or the part of it relative to the current working directory.

You can pass parameters to executable programs just as you can to MCL procedures; the mechanisms are described later in this chapter.

A special form of parameter allows input/output redirection. You can use this either in the program invocation itself, or in the invocation of a calling MCL procedure. The input/output redirection mechanisms are described later in this chapter.

## INTERACTIVE AND BATCH MODES

There are two ways of processing commands, procedures and programs. You can submit them in interactive mode, or by preceding their invocation by 'BM'. Jobs are then placed in a batch queue, and processed according to their priority in the queue.

The MCL procedures containing restart statements can be activated in batch mode (see Chapter 7).

## REDEFINITION OF COMMANDS

MCL commands can be redefined by creating a procedure which contains the command being redefined.

For instance, the COPY command can be redefined by a procedure called SAVE whose contents are:

```
COPY %SOURCE %NEWNAME
COPY %SOURCE %NEWNAME&'.BACK'
```

To copy the file JUPITER to the file SATURN, the user must write:

```
SAVE SOURCE=JUPITER NEWNAME=SATURN
```

and two files, SATURN and SATURN.BACK, will result.

Another interesting example is the redefinition of a non-interactive command into an interactive command. For example, the command WHEREIS could be redefined by the following procedure.

```
READ 'Specify the name of the file:' %FILE;
READ 'Specify the name of the parameter:' %START;
SET %NUL := ' ';
IF %START = %NUL THEN WHEREIS %FILE
                 ELSE WHEREIS %FILE START=%START
```

## SPECIFYING PARAMETERS

The file name of a program or MCL procedure may be followed by a parameter list. A parameter list is a series of one or more parameters separated from the file name and from each other by single spaces.

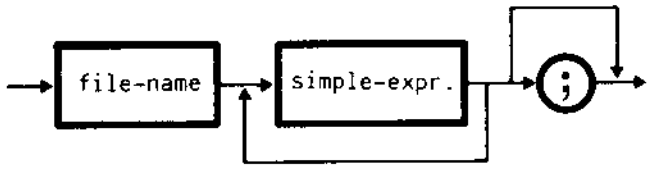
There are two valid types of parameters:

- positional parameters
- keyed parameters.

You can use both sorts of parameters in a single parameter list. Positional and keyed parameters are described on the following pages.

## POSITIONAL PARAMETERS

A positional parameter is a parameter which is identified by a number indicating its position in a sequence of parameters (that is, in a parameter list). The parameter consists of a simple mathematical expression which is resolved at procedure/program activation time to give it a value. A procedure or program with positional parameters may be invoked as follows (from the file where it is stored):




where:

**file-name** is the name of the file containing the procedure or program to be executed

**simple-expr.** obeys the rules for the construction of simple expressions described in Chapter 3.

### Characteristics

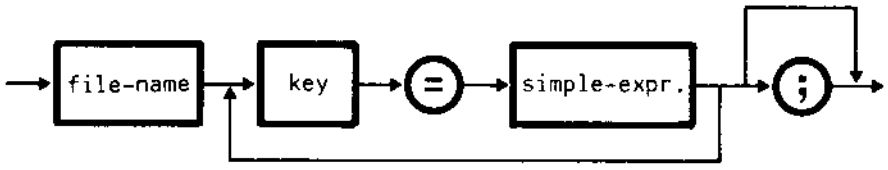
1. In an MCL procedure, the value of the expression is assigned to a variable (which is local to the called procedure) by means of a RDPOS or RDNEXT statement (see Chapter 4 for a description of these statements).

2. Each programming language has its own mechanism similar to MCL's RDPOS and RDNEXT functions, to handle parameter intake.

### Example

```
PROC 37 (%A+12)*%NUMBER %FIRST&'ORD'
```

The argument of a keyed parameter is a simple expression coupled to a key. At procedure or program activation time the expression is evaluated and assigned to a variable name. This variable name is made up of the key and the % prefix. A procedure or program with keyed parameters may be invoked as follows (from the file where it is stored):




where:

**file-name** is the name of the file containing the procedure or program to be executed

**simple-expr.** is a simple expression as described in Chapter 3.

### Characteristics

1. Remember the % prefix for the key within the MCL procedure body. A keyed parameter has the same effect as an assignment statement in the calling procedure.

2. Programming languages each have their own mechanisms to handle parameter intake, and their particular requirements as to intake format.

### Example

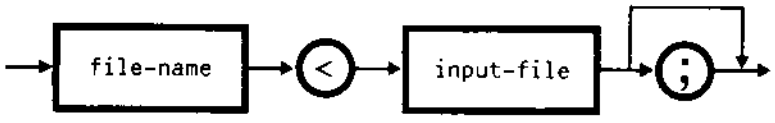
```
PROC FILE=%A3&'.LONDON'
```

## INPUT/OUTPUT REDIRECTION

In the following are described the mechanisms for I/O redirection of MCL procedures and programs, if you don't want to use the standard input from keyboard and the standard output from video.

### CHANGING STANDARD INPUT

To specify a standard input other than the terminal keyboard, use the following syntactical form in the procedure or program invocation:




where:

**file-name** is the name of the file containing the procedure or program to be executed

< indicates that until another instruction of this type is input the specified input-file will be used as standard input

**input-file** is either the path name of an existing byte-stream file or a local name which has previously been connected to a global name; it may also be a device name; device names accepted are as follows:

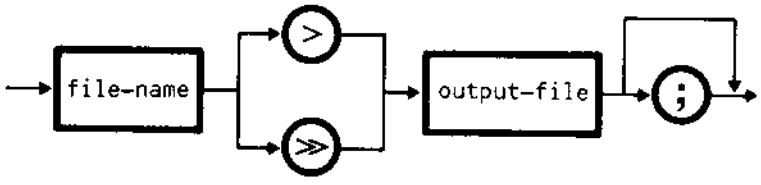
/DEV/TTYA : keyboard of the 1st work station
/DEV/TTYB : keyboard of the 2nd work station
/DEV/TTYC : keyboard of the 3rd work station
/DEV/TTYD : keyboard of the 4th work station
.
.
.

#### Characteristics

1. If you use the standard input redirection in the invocation of an MCL procedure, the redirection only applies to procedures and programs nested within that procedure. The standard input of the outermost procedure is always the keyboard.

2. You cannot redirect input to an MCL built-in statement.

To specify a standard output other than the terminal video, use the following syntactical form in the procedure or program invocation:




where:

**file-name** is the name of the file containing the procedure or program to be run

**>** indicates that the redirected output is to overwrite the previous contents (if any) of the output file

**>>** indicates that the redirected output is to be appended to the previous contents of the output file

**output-file** is either a device name or the path name of a byte-stream file; if the file you identify does not already exist, it is created; it may also be a local name which has previously been connected to a global name (see the command CONN); device names accepted are as follows:

```
/DEV/TTYA : video of the 1st work station
/DEV/TTYB : video of the 2nd work station
/DEV/TTYC : video of the 3rd work station
/DEV/TTYD : video of the 4th work station
 .
 .
 .
/DEV/SYSPRT1 : 1st system printer
/DEV/SYSPRT2 : 2nd system printer
 .
 .
 .
```

**Characteristics**

1. If you use the standard output redirection in the invocation of an MCL procedure, the redirection only applies to procedures and programs nested within that procedure.

2. You cannot redirect output from an MCL built-in statement.

## COMPLETION CODES

To inform the user of the result of each Shell command (including the commands for the system administrator and those for system installation and maintenance) or more generally of a requested activity, and on the basis of this, decide which operation to execute, Shell emits a completion code. This is loaded in the MCL variable "%STATUS" that can assume the following values:

0 Execution correctly terminated; there is no information for the user.

1 At least one warning has occurred during execution. Information about this can be found in the MCL variable "%RET".

2 At least one error has occurred in or outside the activity. Execution is interrupted and further information, corresponding to the error, can be found in the MCL variable "%RET".

127 Execution terminated correctly and further information (results, etc.) has been loaded in the MCL variable %RET.

128 Request for shutdown programmed from an interactive terminal (handled by Grandpa, not Shell).

129 Request for shutdown and automatic switch off (handled by Grandpa, not Shell).

If a warning message is emitted and an error occurs, the user is only informed of the most serious one (the error, in this case).

The values of the variables %STATUS and %RET can be set explicitly, being careful not to lose any information loaded by the system.

A built-in function, provided by the programming language being used, is available to the user for loading information in the variable %RET.

Completion codes 0, 1 and 2 are usually set up by all the compilers, run-time supports, linkers, etc. For any discrepancy that may occur see the relative manuals.

# 6. EDITING MCL PROCEDURES

To create a repeatable MCL procedure write your MCL statements into a new file by means of the system editor. Group your procedures in a suitable subdirectory and use meaningful file names whenever possible. Make sure that any parameters you pass are appropriate to the procedure and correctly ordered.

To run a syntactic check on a new procedure without executing it, use the MCL built-in statement EDPARSE. This is described in Chapter 4. You can also use EDPARSE to create new files, but it only accepts one input line each time it is invoked.

The effectiveness of an MCL procedure can best be confirmed at run time. You can use the RETURN, LENGTH, SHVAR, SUBSTR and WRITE statements to report on the working of your procedures.

# 7. RESTART MECHANISM

This chapter describes the restart mechanism for MCL procedures when there is a system crash.

The mechanism is activated automatically if it is requested using the ONRST...ENDRST instruction, straight after system initialisation.

Shell creates a file for each user connected to the system, in which is stored the status of the environment at the time of the crash. The name of this file is:

IR$user-name

and it resides in the home directory, which is active straight after user login. This contains the procedure information necessary to restore this procedure after a system crash:

- procedure path name
- procedure input parameters
- name of current directory when the procedure was activated
- offset of the last ONRST...ENDRST instruction encountered in the procedure containing the restart instruction
- all the global and local variables initialised when requesting restart with the ONRST...ENDRST instruction.

The file created by Shell is removed if the procedure ends correctly. If a file IR$user-name exists at initialisation, it means that the system had crashed. The interrupted procedure is then restarted automatically.

The file is also removed and the Shell environment reset when the procedure needing to be restarted is killed.

For procedures that are to be executed in batch mode Shell creates the file:

BR$job-number

in the user working directory. If a crash occurs when a job is being executed in batch mode, it is appended to the job queue during initialisation, and is restored when activated again.

# 8. MEMORY UTILISATION

This chapter describes the internal data structure and memory utilisation of Shell. This allows the user to determine the amount of memory which will be used by an MCL procedure.

## DATA STRUCTURE

Shell uses the following data areas:

- STRING SPACE: this contains all the character strings used by Shell. It is divided into two parts: one containing the names and values of variables, the other containing strings, both allocated and deallocated dynamically.
- SYMBOL TABLE: this contains the pointers to the names and values of all MCL variables.
- VMSTACK: this is a static stack which contains intermediate value of logical and arithmetic operations, and also return pointers for nested procedures.
- CODE AREA: this consists of the intermediate codes used by Shell.

## AVAILABILITY

Memory space for an MCL procedure is allocated dynamically, in blocks of 4K bytes.

Occupation of each data structure for each new memory allocation is organised as follows:

| dynamic allocation | |
|---|---|
| STRING SPACE | strings: 1k bytes<br>variables: 1k bytes |
| CODE AREA | 256 intermediate instructions: 1k bytes |

| static allocation | |
|---|---|
| SYMBOL TABLE | 225 variables |
| VMSTACK | 195 elements |

The two data areas "STRING SPACE" and "CODE AREA" may occupy a total of 64K bytes of memory.

## UTILISATION

The actual utilisation of the available memory depends on the content of the procedure being executed.

### STRING SPACE

Each character string uses one byte per character. The values of variables occupy a variable number of bytes depending on the length of the string needed to represent the value.

### SYMBOL TABLE

In addition to the pointers to all variables, each case of nesting uses eight elements for the pre-defined variables. The parameters used during procedure calls are also stored in this area.

### VMSTACK

Ten elements of this area are used for each case of nesting or procedure call. Utilisation during the evaluation of expressions varies.

### CODE AREA

The principal MCL keywords occupy memory space as follows:

| KEYWORD | No. bytes |
|---|---|
| SET | 2 |
| IF | 4 |
| WHILE | 4 |
| FOR | 9 |
| Procedures | 3 each |

Operational parameters occupy memory space as follows:

| PARAMETER | No. bytes |
|---|---|
| Positional | 2 |
| Keyed | 3 |
| Logical or arithmetic | 2 |

In addition, two instructions are used at each exit of a procedure.

## DEALLOCATION

When a procedure is called, its instructions are read one at a time and the relative values are allocated in the string space, in the symbol table and in the code area.

The instructions are interpreted one by one and then deallocated.

# A. MCL ERROR MESSAGES

When Shell discovers an error condition, either through the EDPARSE program or at run time, it puts an error number on the screen, along with a brief message of explanation. For example:

MCL ERROR 3 : I/O ERROR

This appendix contains a detailed list of all such MCL errors.

MCL ERROR 1 : NOT FOUND

There is no such command.

MCL ERROR 2 : NOT EXEC FILE

The specified file is not executable.

MCL ERROR 3 : I/O ERROR

A system error occurred when reading a file. The error is detected by the file system.

MCL ERROR 4 : I/O ERROR

A system error occurred when writing a file. The error is detected by the file system.

MCL ERROR 5 : STRING TOO LONG

The specified input string contains more than 160 characters. This error can also be displayed when the path name of the specified file exceeds 60 characters.

MCL ERROR 6 : BAD INCLUSION

The inclusion options cannot be nested.

MCL ERROR 7 : BAD INCLUSION

The name specified for the inclusion option does not exist.

MCL ERROR 8 : CMD TOO LONG

A command longer than 160 characters has been entered.

MCL ERROR 9 : FILE IN USE

An attempt has been made to execute an MCL procedure already opened in exclusive mode.

---

MCL ERROR 10 : I/O ERROR

A system error occurred during a file system operation.

---

MCL ERROR 11 : SYNTAX ERROR

The word THEN is missing from an input line. The keyword may have been left out, or entered wrongly.

---

MCL ERROR 12 : SYNTAX ERROR

The variable name has not been entered straight after the keyword SET in an input line.

---

MCL ERROR 13 : SYNTAX ERROR

The characters ":=" are missing from the "SET variable name" command in an input line.

---

MCL ERROR 14 : SYNTAX ERROR

The closing bracket ")" is missing from an input line.

---

MCL ERROR 15 : SYNTAX ERROR

Carriage return has been pressed before the whole command has been entered (for instance the quotes ending a constant definition have not been written).

---

MCL ERROR 16 : ILLEGAL PARAMETERS

The keyword of a keyed parameter must be a string.

---

MCL ERROR 17 : SYNTAX ERROR

The wrong operand has been entered in an arithmetical or logical expression.

---

MCL ERROR 18 : ILLEGAL OPERATION

Too many files have been connected to SPOOL classes. The commands CONN and CONSP only allow three connections to be made to SPOOL classes.

---

MCL ERROR 19 : ILLEGAL INPUT

There are illegal characters in an input line. This error message can also be

emitted when a byte-stream file whose name has been specified for execution does not contain an MCL procedure or an executable program.

---

MCL ERROR 20 : SYNTAX ERROR

A semi-colon ";", which is the MCL separator, is missing from an input line.

---

MCL ERROR 21 : SYNTAX ERROR

An instruction has been entered wrongly in an input line. Check the syntax of the instruction.

---

MCL ERROR 22 : SYNTAX ERROR

An incorrect option has been given as input to a function, or the metacharacter "" is not followed by a character string.

---

MCL ERROR 23 : SYNTAX ERROR

The keyword DO is missing from an input line. The keyword may have been left out or entered wrongly.

---

MCL ERROR 24 : SYNTAX ERROR

In an arithmetical expression an operand contains alphanumeric characters, whilst the string should be numeric.

---

MCL ERROR 25 : SYNTAX ERROR

In an IF...THEN...ELSE statement, an invalid condition has been specified.

---

MCL ERROR 26 : SYNTAX ERROR

The wrong number of parameters has been specified in one of the following commands: LENGTH, SHVAR, SUBSTR, WRITE, SETWDIR, EDPARSE, RDPOS, RDNEXT, ECHO, CONN, CONSP, DISCON, READ, SHNAME.

---

MCL ERROR 27 : STRING TOO LONG

The string resulting from linking variables is longer than 160 characters.

---

MCL ERROR 28 : ILLEGAL PARAMETERS

The local name specified in the command CONN has already been connected to a global name.

---

MCL ERROR 29 : aaa...a NOT FOUND

The local name "aa...a" specified in the command DISCON is not one of the local names connected.

---

MCL ERROR 30 : ILLEGAL OPERATION

The SPOOL class specified for connection with a local name with the command CONN or CONSP does not exist. The existing classes are:

- SPOOL1 - SPOOL16 (for CONN and CONSP)
- SP1NxMy - SP16NxMy (where "x" is the network number and "y" is the machine number, for CONSP only).

---

MCL ERROR 31 : ILLEGAL PARAMETERS

A variable is missing from a command parameter. This error may occur during execution of the following commands: SHVAR, RDPOS, RDNEXT, READ.

---

MCL ERROR 32 : INVALID PATHNAME

The global name specified in the command CONN has an invalid path name.

---

MCL ERROR 33 : ILLEGAL PARAMETERS

The result of an elementary expression is alphabetical rather than numeric. This error may occur in the SUBSTR and RDPOS commands.

---

MCL ERROR 34 : ILLEGAL PARAMETERS

In the SUBSTR command, the parameter that specifies the position of the first character in the string is negative or is longer than the string; or the parameter specifying the number of characters making up the string is negative.

---

MCL ERROR 35 : ILLEGAL OPERATION

The command LOGOUT has been entered, but not executed, because a procedure or program is still waiting. The user must end (KILL command) or restart (RESUME command) the procedure or program suspended.

---

MCL ERROR 36 : ILLEGAL OPERATION

The RESUME command has been entered, but there are no suspended programs or procedures.

---

MCL ERROR 37 : ILLEGAL OPERATION

The LOGOUT and RESUME commands cannot be inserted in an MCL procedure. They can be activated only interactively.

---

MCL ERROR 38 : ILLEGAL OPERATION

The MCL procedure or program being run cannot be suspended as two MCL procedures or programs have been suspended already.

---

MCL ERROR 39 : INVALID PATHNAME

The path name specified in the SETWDIR command is invalid.

---

MCL ERROR 40 : INVALID PATHNAME

The file specified in the EDPARSE command is not a byte-stream file.

---

MCL ERROR 41 : ILLEGAL OPERATION

The command BM cannot be nested.

---

MCL ERROR 42 : ILLEGAL OPERATION

The batch environment is not available as it has not been started up in the initialisation phase (Grandpa).

---

MCL ERROR 43 : LACK OF MEMORY

There is an overflow in the string table.

---

MCL ERROR 44 : LACK OF MEMORY

Too many variables have been created, and an overflow occurred in the symbol table.

---

MCL ERROR 45 : LACK OF MEMORY

The MCL procedure activated has too many nested procedures for the available space.

---

MCL ERROR 46 : LACK OF MEMORY

There is an overflow in the MCL data area.

---

MCL ERROR 47 : ILLEGAL OPERATION

A standard system connection cannot be disconnected.

---

MCL ERROR 48 : ILLEGAL OPERATION

The EDPARSE command cannot be used in an MCL procedure.

---

MCL ERROR 49 : BAD INCLUSION

It is not possible to include the name of a file which is an argument to the command EDPARSE.

---

MCL ERROR 50 : SYNTAX ERROR

An EXIT must be included in a FOR or WHILE statement.

---

MCL ERROR 51 : PRG CAN'T RUN

It is not possible to create a new family, because there is no more space in the system table.

---

MCL ERROR 52 : PRG CAN'T RUN

The program cannot be loaded for execution because there is not enough memory, or because the program file is illegible. This error comes from the LOAD primitive.

---

MCL ERROR 53 : PRG CAN'T RUN

The program cannot be run because an attempt has been made to load it in a segment already used, or which does not belong to the family.

---

MCL ERROR 54 : PRG CAN'T RUN

The program cannot be loaded because the type and attributes of the segment in which it should be loaded have been defined incorrectly, its length is wrong or the PMM does not recognise that the program is executable.

---

MCL ERROR 55 : PRG CAN'T RUN

The program cannot be loaded because the program table has overflown.

---

MCL ERROR 56 : PRG CAN'T RUN

The program cannot be started because the process table has overflown.

---

MCL ERROR 57 : PRG CAN'T RUN

A program suspended with the command SUSPEND cannot be restarted.

---

MCL ERROR 58 : SYNTAX ERROR

A TO is missing in a FOR statement.

---

MCL ERROR 59 : BAD STANDARD I/O

Redirection of an operation has been specified incorrectly: the name given after the redirection operand does not correspond to a byte-stream file.

---

MCL ERROR 60 : BAD STANDARD I/O

The redirection is ambiguous, or the name given after the redirection operand does not exist.

---

MCL ERROR 61 : SYNTAX ERROR

The search command character "?" cannot be used here.

---

MCL ERROR 62 : SYNTAX ERROR

The repeat character "!" cannot be used here.

---

MCL ERROR 63 : NOT FOUND

The string entered is not recognised as a file system object.

---

MCL ERROR 64 : ILLEGAL OPERATION

Am empty job has been submitted for batch processing.

---

MCL ERROR 65 : STRING TOO LONG

The path name of the work directory is too long.

---

MCL ERROR 66 : INVALID ID

The specified directory or file does not exist.

---

MCL ERROR 67 : ILLEGAL PARAMETERS

The parameter "local name" of the commands CONN or CONSP cannot start with the character "/".

---

MCL ERROR 68 : CMD TOO LONG

Too many arguments have been given to the program procedure started.

---

MCL ERROR 69 : ILLEGAL OPERATION

Tracing has been used incorrectly.

---

MCL ERROR 70 : ILLEGAL OPERATION

The keywords ONRST...ENDRST have been used incorrectly. The user must remember that they can only be used at the first level of nesting in a procedure and not in an interactive environment.

---

MCL ERROR 71 : NO MATCH

The file name specified using one or more "?" characters does not correspond to an actual file.

---

MCL ERROR 72 : AMBIGUOUS CMD

A command has been specified using one or more characters "?" and there is more than one correspondence.

---

MCL ERROR 73 : INVALID OPERATION

Private buffers cannot be assigned, for one of the following reasons:

- the number of private buffers to assign exceeds the number defined in the configuration file
- the number of private buffers to assign is greater than that of available buffers
- the private buffers have already been assigned to the same file by another user.

This message is displayed by the CONN statement.

---

MCL ERROR 74 : I/O ERROR

Synchronous I/O cannot be requested. This message is displayed by the CONN command.

---

MCL ERROR 75 : LACK OF MEMORY

Overflow in the table containing the list of commands entered.

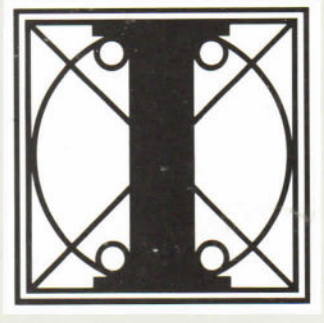
---

MCL ERROR 76 : ILLEGAL OPERATION

The access rights do not allow the operation requested on the specified file system item.

---

Printed in Italy



olivetti