MP/OS Utilities Reference

DataGeneral
Data General Corporation, Westboro, Massachusetts 01581

Notice

Data General Corporation (DGC) has prepared this document for use by DGC personnel, licensees, and customers. The information contained herein is the property of DGC and shall not be reproduced in whole or in part without DGC prior written approval.

DGC reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult DGC to determine whether any such changes have been made.

THE TERMS AND CONDITIONS GOVERNING THE SALE OF DGC HARDWARE PRODUCTS AND THE LICENSING OF DGC SOFTWARE CONSIST SOLELY OF THOSE SET FORTH IN THE WRITTEN CONTRACTS BETWEEN DGC AND ITS CUSTOMERS. NO REPRESENTATION OR OTHER AFFIRMATION OF FACT CONTAINED IN THIS DOCUMENT INCLUDING BUT NOT LIMITED TO STATEMENTS REGARDING CAPACITY, RESPONSE-TIME PERFORMANCE, SUITABILITY FOR USE OR PERFORMANCE OF PRODUCTS DESCRIBED HEREIN SHALL BE DEEMED TO BE A WARRANTY BY DGC FOR ANY PURPOSE, OR GIVE RISE TO ANY LIABILITY OF DGC WHATSOEVER.

Ordering No. 093-400002

Rev. 01, April 1981

Preface

Organization of the Manual

This book is intended to serve the needs of programmers who want to use the utilities available with the MP/OS operating system.

The book is divided into nine sections, each of which describes one of the MP/OS utilities in detail:

Section 1 describes the basic functions of the *Command Line Interpreter*, and goes on to discuss the powerful macro facility. Finally, all the CLI commands are described in detail in a "dictionary" chapter.

Section 2 starts by discussing the types of operation you can perform with the *Speed Text Editor*. It goes on to introduce the basic Speed commands and show some sample editing sessions. Finally, all the Speed commands are described in a "dictionary" chapter.

Section 3 describes the operating instructions for the *Macroassembler*, and then goes on to describe macros and pseudo ops in detail. Finally, all the pseudo ops are presented in a "dictionary" chapter.

Section 4 discusses in detail the functions of the *Binder and Library Editor*.

Section 5 introduces the reader to the *Symbolic Debugger*, and goes on to describe the various categories of Debugger commands. Finally details of all the commands are presented in a "dictionary" chapter.

Section 6 describes *Disk Initialization* using the DINIT and MDINIT utilities.

Section 7 introduces the *Move Utility*, and goes on to describe how to use the utility to create back ups.

Section 8 describes, by the means of examples, how you use the *AOS File Transfer Utility*.

Section 9 discusses how you can use the *File Display and Comparison Utility*.

The appendices at the end of the manual list the error messages returned by the various utilities.

Related Manuals

The list that follows gives a brief description of each of the other manuals which describe Microproducts and the MP/OS system.

- *An Introduction to Microproducts and Micron* (DG No. 069-400000) describes the hardware and software in general terms, to give an overview of your MP/Computer and its capabilities.
- *Microproducts Hardware Systems Reference* (DG No. 014-000636) gives a detailed functional description of the Microproducts line of microcomputers and related peripherals, board by board.
- *Learning to Use the MP/OS Operating System* (DG No. 093-400000) should be read by anyone who has never used an MP/Computer. It introduces the MP/OS file system and the Command Line Interpreter. A console session gives you step by step hands-on experience with your new MP/Computer.
- *MP/OS Assembly Language Programmer's Reference* (DG No. 093-400001) is the main source of information for the assembly language programmer. It describes the instruction sets of MP/Computers in detail, and gives details of the Macroassembler and operating system.
- *MP/Fortran IV Programmers' Reference* (DG No. 093-400004) covers all of the features of this powerful high level language.
- *MP/Pascal Programmers' Reference* (DG No. 093-400003) describes Data General's extended

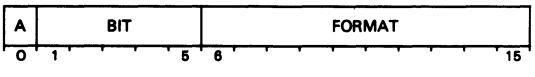
version of Pascal.

Typesetting Conventions

Throughout this manual we use the following conventions to show instruction formats:

COMMAND	We use bold face and uppercase letters to indicate the instruction mnemonics. You code them into your program exactly as they appear.
argument	We use italics and lower case letters to indicate that a particular instruction takes an argument. In your program, you must replace this symbol with the exact code for the argument you need.
[optional]	Brackets denote an optional argument. (Command switches appear in this format as well.) If you decide to use this argument or switch, do not write the brackets into your code: they only set off the choice.
[...optional]	Denotes a number of optional arguments.
arg1 | *arg2*	Denotes *either arg1 or arg2.*
RESPONSE	We use upper case italics to show any response that the program may return.
Example	We use sans serif for all programming examples.

In addition, we use the following special symbols:



This diagram shows the arrangement of the 16 bits in an instruction. The diagram is always divided into 16 boxes, numbered 0 to 15.

<↓>
represents a New-line character.

CTRL means that you should depress and hold the Control key while you press the character following the CTRL.

Contents

Section 1
The Command Line Interpreter

Chapter 1
Introduction

Functions of the CLI

The Command Line Interpreter (CLI) is your primary interface with the MP/OS system. By typing in CLI commands at a console, you can communicate with the operating system. The CLI allows you to:

- Manage the MP/OS file system
- Invoke other MP/OS utilities
- Invoke user programs
- Control the system environment

In addition to regular CLI commands, you can use the macro feature to create your own commands.

Managing Files

CLI commands allow such operations as creating, deleting, renaming, and copying files. You can obtain information about the status of a file and can also print or display the contents of a file (depending on whether you have a hard-copy terminal or a display terminal).

Invoking Other Utilities

The CLI also lets you invoke other MP/OS utilities:

Speed
Macroassembler
Binder
Library Editor
Symbolic Debugger
Disk Initializer
Move Utility
File Compare and Display Utility
MP/Pascal
MP/Fortran IV

To use any of the above programs or one of your own programs type either **XEQ** or **EXECUTE** followed by the name of the program you want. When you finish using the program you are returned to the CLI.

Managing the System Environment

Certain features of the system environment can be set with CLI commands. For example, you can set the current working directory, the searchlist, and the characteristics of devices.

Organization of this Section

This section of the manual is organized as follows:

Chapter 1 - Introduction
Defines the CLI and explains the organization of the manual.

Chapter 2 - The File System
Describes the MP/OS file system and explains how to reference and manipulate files.

Chapter 3 - Console Control Characters
Shows you how to control what is happening at the console with console control characters and sequences.

Chapter 4 - CLI Syntax
Explains CLI command formats.

Chapter 5 - The Macro Facility
Explains the purpose of macros and shows you how to create and use them.

Chapter 6 - Summary of CLI Commands
Summarizes all the CLI commands according to their function.

Chapter 7 - Command Dictionary
Provides a detailed explanation of each CLI command.

Chapter 2
The File System

Files and Directories

A file is either a device (e.g., @LPT, @DPD0) or a collection of data on a device. It has a filename to identify it. Filenames can be up to 15 characters long and consist of the following characters:

a through z
A through Z
0 through 9
? $. _

> **NOTE:** *The system does not distinguish between upper and lower case alphabetics.*

You may use any conventions you like when naming files, but you should be aware that the system follows these conventions when naming files:

- Assembly language source files end in **.SR**
- Object files end in **.OB**
- Executable program files end in **.PR**
- Permanent symbol table files end in **.PS**
- Symbol table files end in **.ST**
- MP/Fortran IV source files end in **.FR**
- MP/Pascal source files end in **.PAS**

Files are grouped into directories. A directory is simply a file which can contain other files. It can also contain other directories (sub-directories) which in turn may contain other files and/or sub-directories. A directory containing sub-directories is a parent directory. Directories can nest to any level, and the result is a hierarchical file system. (The level of nesting is in fact only limited by the 127 character limit on pathnames.)

A directory has information about all the files and sub-directories (if any) it contains. It has no information about files contained in some other directory in the system.

The MP/OS system's hierarchical (tree-structured) file system is organized as follows: the highest directory in the system is the device directory which has the name @. This directory is not actually a disk file, but a table in the system's memory. The device directory contains the filenames of all the input/output devices in the system. These filenames are preceded by @ (i.e., @LPT, @DPD0, etc.). Non-disk devices do not contain directories. Disk devices, on the other hand, can contain directories and files in a tree structure. The tree structure originates in a root directory, which is referenced by the disk device name followed by a colon, (e.g., @DPD0:). The colon differentiates the root directory from the device itself.

Every MP/OS system has one disk device which is designated as the system master device. If you have only one disk device in your system, it will be the system master device by default.

The system master device is the device from which the system was booted. It may also contain directories and files you create. You have the option of referring to the system master device with a colon (:) instead of @*devicename:*. In our examples we will assume that your files are on the system master device.

Figures 2.1 and 2.2 are examples of hierarchical file systems under the MP/OS system. Figure 2.1 has only one disk device, @DPD0, which, by default is the system master device. Figure 2.2 has two disk devices. @DPD0 and @DPD1. @DPD0 is designated as the system master device. Both @DPD0 and @DPD1 contain user files and directories.

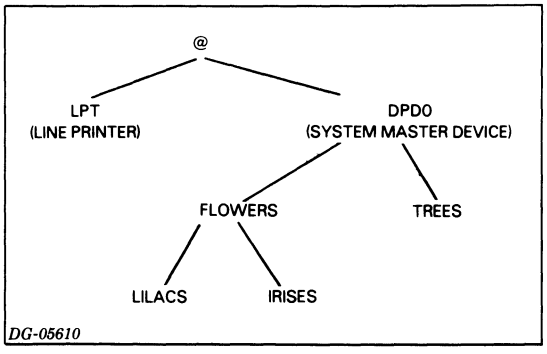


Figure 2.1

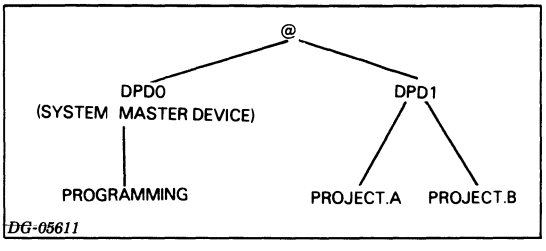


Figure 2.2

File Types and Attributes

We have discussed one type of file, the directory. Some other file types are as follows:

Character device
Line printer
Directory device
Program file
Break file
Push or swap file
Range of system defined data files
Range of user defined data files

Files can also have certain attributes:

P: This file is permanent and cannot be deleted.
R: This file is read-protected.
W: This file is write-protected.
A: This file's attributes cannot be changed.

Attributes are set by the system, never by the user.

Referencing Files

Pathnames

A pathname represents the unique path through the file system to a specific file. If you are referencing a file on the system master device, the pathname begins with the system master device name (e.g., @DPD0:), followed by one or more filenames separated by colons. All the filenames except for the last one must be the names of directories, and each directory named must be a sub-directory of the preceding one. Pathnames can be up to 127 characters long. So, using the example of the tree in Figure 2-1, the pathname to IRISES is **@DPD0:FLOWERS:IRISES**. (You could have abbreviated the name of the root directory of the system master device to a colon(:) instead of @DPD0:, giving you this pathname **:FLOWERS:IRISES**.)

If you are referencing a file on a device other than the system master device, the pathname always begins with @. Therefore, looking at the directory tree in Figure 2.2, the pathname to PROJECT.A is **@DPD1:PROJECT.A**.

Since pathnames can get long and cumbersome, the MP/OS operating system provides two other ways of referencing files: the working directory and the searchlist.

Working Directories

The working directory is, as its name suggests, the directory you are currently "in". You can reference any file in the working directory with just a filename instead of the whole pathname.

Whenever you reference a file simply with a filename, the MP/OS system assumes you are referring to a file in the working directory. If you need to reference a file in the parent directory of the working directory, you can use the prefix ↑ (circumflex or up-arrow, depending on your terminal). You can use more than one of these (if you are moving up more than one level), but they must always precede the filename. In addition, if you want to reference a file in the sub-tree of the working directory, you can use a partial pathname.

Partial Pathnames

A partial pathname begins with a sub-directory of the working directory and is followed by one or more filenames, ending with the file you are referencing. Using the example of the directory tree in Figure 2.3, suppose the working directory is NYC and you want THEATRES as the working directory. Instead of using a full pathname you can use a partial pathname with the CLI **DIRECTORY** command:

```
) DIRECTORY MANHATTAN:THEATRES<↓>
```

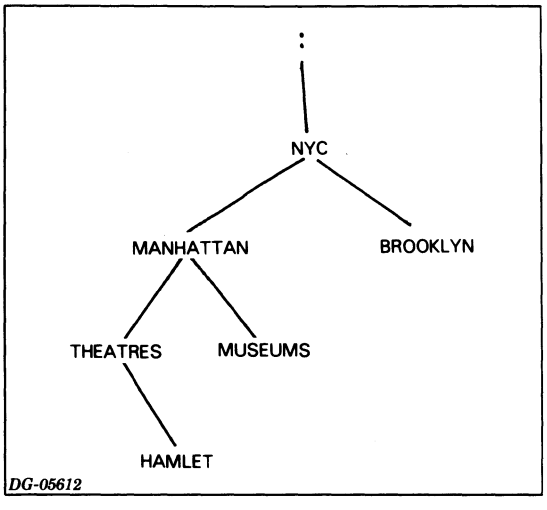


Figure 2.3

Searchlists

A searchlist is an ordered list of directories you want searched any time a file referenced with just a filename or a partial pathname is not in the working directory. In other words, the working directory is searched first, and if the file is not there, the system goes through the searchlist. If, however, you want to restrict the search for a file to the working directory, use the prefix "=" before the filename.

Templates

A filename template is any legal filename plus one of the special template characters. These special characters are as follow:

asterisk (*)
hyphen (-)
plus sign (+)
back slash (\)

Template characters may be used with certain CLI commands as a short-cut when providing arguments. The template character tells the CLI to access all files meeting the criteria specified by the template character. Thus the CLI interprets each character in the filename template literally except for the template character. The template characters are interpreted as follows:

- An asterisk (*) results in any single character except for a period, null character, or null string being matched.
- A hyphen (-) results in any character string (including the null string) that does not contain a period being matched.
- A plus sign (+) results in every character string, including those containing periods and the null string being matched.
- A back slash (\) is used in conjunction with another template character to limit the set of filenames to be matched. In other words, the CLI accesses each file meeting the specifications of the template *except* for the file or template character following the back slash.

Figure 2.4 illustrates the relationship between the templates discussed above and specific filenames.

Template	Filenames matched
FILEA	FILEA
FILE*	FILEA, FILEB
FILE-	FILEA, FILEAB, FILEB, FILEXY
FILE+	FILEA, FILEAB, FILEB, FILEXY, FILE.1, FILE.2
FILE+\FILEA	FILEB, FILEXY, FILE.1, FILE.2, FILEAB
CPU.-	CPU.1, CPU.2
CPU+	CPU.1, CPU.2, CPUADDENDA
+	ALL FILENAMES

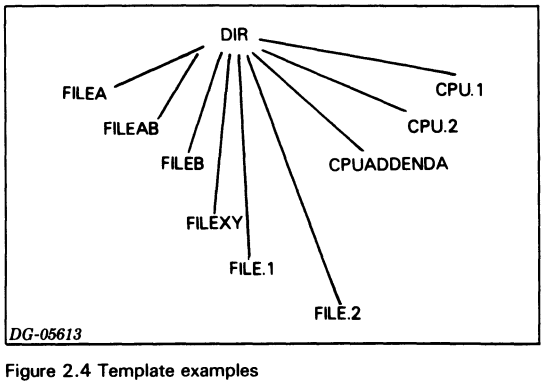


Figure 2.4 Template examples

File Manipulation

Creating Directories and Files

We have already described the file system under the MP/OS system. Using the CLI **CREATE** command you can create files and directories, thus producing the hierarchical file system. The structure of the hierarchy is up to you: you can have many directories with the same parent directory, multiple levels of directories, or a combination of the two.

Changing the Working Directory

Using the example in Figure 2.5, let us suppose the working directory is FICTION and you want CLARISSA as the working directory. Use the CLI **DIRECTORY** command, followed by a partial pathname:

```
) DIR RICHARDSON:TRAGEDY:CLARISSA<↓>
```

> **NOTE: DIRECTORY** *can be abbreviated to* **DIR**.

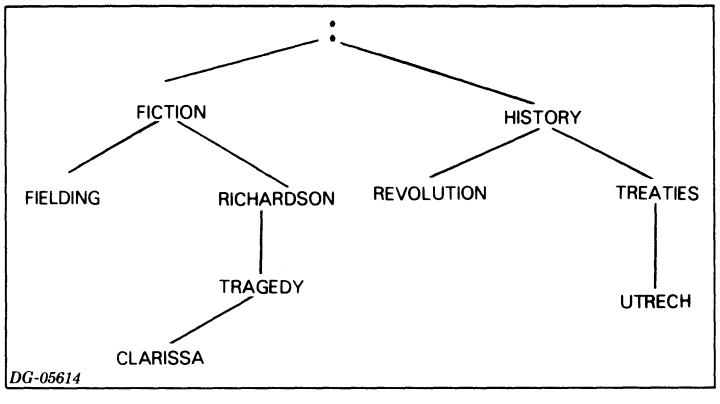


Figure 2.5

If, after working in CLARISSA you wanted TRAGEDY as the working directory, you would use the circumflex (or up arrow) to accomplish this:

```
) DIR ↑<↓>
```

If you then wanted UTRECHT as the working directory, you could either go up to the root directory and then down:

```
) DIR ↑↑↑HISTORY:TREATIES:UTRECHT<↓>
```

or specify a pathname:

```
) DIR :HISTORY:TREATIES:UTRECHT<↓>
```

Copying Files

The CLI **COPY** command allows you to make a copy of one or more files to a destination file. If the destination file already exists, you can either delete and recreate it prior to the copy operation, or you can append the source file(s) to it, thus effectively combining several files. For details see Chapter 7, "CLI Command Dictionary".

Renaming Files

No two files in the same directory can have the same name, although files in different directories can. The CLI, however, allows you to rename files so that if, for instance, you are copying a file called INSTRUCTIONS to another directory which already has a file called INSTRUCTIONS, you can rename one of the files so that you can have both in the same directory. The **RENAME** command also lets you graft directories and files from one part of the directory tree to another by specifying pathnames in the argument list. For example, using the illustration in Figure 2.5, TREATIES can be made a sub-directory of FIELDING. The command line you would use to accomplish this, and the resulting directory tree are shown in Figure 2.5.

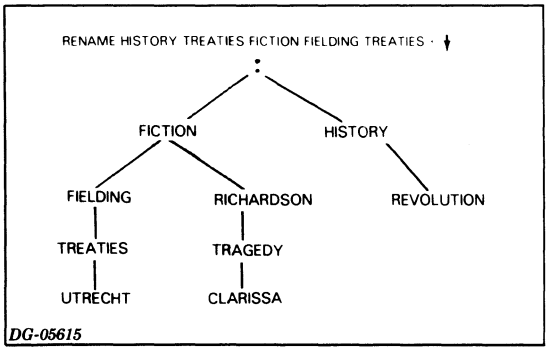


Figure 2.6

> **NOTE:** *The* **RENAME** *command treats partial pathnames as if they start in the working directory.*

Deleting Files

If at any point you decide you no longer need a directory or file, the CLI allows you to delete it. You can use the **DELETE** command to delete files or empty directories. In conjunction with the **/DIR** switch you can delete any sub-trees the specified directory contains.

Listing Files

Should you wish to find out which files your working directory contains, you can use the **FILESTATUS** command, which results in the filenames being listed on the console.

Displaying Files

If you want to display the contents of a file on your console, you can use the CLI **TYPE** command. This does not let you change the contents of a file; it just lets you examine it.

Chapter 3
Console Control Characters

Control Characters

When working at the console you may find you either want to change something you have typed or stop what is happening at the console. For example, you can use the Delete or Rubout key (depending on your console) to correct typographical errors. Delete/Rubout acts as a destructive backspace, deleting one character at a time. In addition, you can use certain control characters to affect what is happening at the console. Control characters are transmitted when you press the CTRL key and then press some other key while still holding down CTRL. We represent them as **CTRL-***x* where *x* is the other key. Control characters are usually echoed as ↑ *x*.

The following are console control characters:

CTRL-C signifies the start of a control sequence.
CTRL-D indicates the end of a file. Used to terminate file input from the console.
CTRL-Q cancels the effect of **CTRL-S**: output to the console is resumed.
CTRL-S suspends output to the console. You may want to use **CTRL-S** if the program you are running is generating a lot of output and you want to stop and examine it at some point.

> **NOTE:** *If you type* **CTRL-S** *by accident, you may think that the system is dead because nothing you do will show results at the console. When this happens, type* **CTRL-Q**. *If* **CTRL-S** *was the cause of the problem,* **CTRL-Q** *will negate its effect.*

CTRL-T replaces the current line. This is particularly useful if you have done several Deletes at a hard copy terminal.

CTRL-U erases the current command line. This saves you typing a series of Deletes if you change your mind about entering a command.

Control Sequences

There are also control sequences you can use. All control sequences consist of **CTRL-C** followed by another control character:

CTRL-C CTRL-A is program-dependent. If the program is the CLI, **CTRL-C CTRL-A** interrupts the CLI. You get a console interrupt error message, followed by a prompt (in the form of a right parenthesis) on the next line.
CTRL-C CTRL-B immediately terminates the program that is running and returns you to the parent program. If the program is the root CLI, it is refreshed and re-invoked.
CTRL-C CTRL-E terminates the current program and creates a break file containing the memory image of the terminated program. You cannot use this command to terminate the root CLI. This sequence is primarily useful in debugging a program.

Chapter 4
CLI Syntax

The CLI prompt is a right parenthesis followed by a space ") ". It indicates that the CLI is ready to accept commands.

CLI command lines consist of a command alone or of a command followed by one or more arguments. An argument may consist of a filename, program name, etc. Some commands require arguments, others allow them, and still others do not accept arguments.

Delimiters and Terminators

If the command line has one or more arguments, you must separate the command from the argument and the arguments (if more than one) from each other. To do so you can use the following delimiters:

- One or more spaces.
- One or more tabs.
- A single comma.
- Any combination of spaces, tabs, and single commas.

Furthermore, a command is not transmitted to the CLI until you terminate it properly. You can use the following as terminators:

New-line
Form Feed
Carriage Return

So, for example, depending on which delimiters you used, a command line could look like one of the following:

```
) CREATE FLOWERS<↓>
```

```
) CREATE,FLOWERS<↓>
```

```
) CREATE  FLOWERS<↓>
```

Switches

You can modify certain CLI commands by using switches. A switch may consist of one or more characters. Switches can either be *simple* or *keyword*.

A simple switch has the format:

/switchname

For example, you can modify the **COPY** command so that the contents of one file are appended to another file. To do so, you would use the **/A** switch:

```
) COPY/A TRIG COSINE<↓>
```

A keyword switch has the format:

/keyword=value

For example, you can create a file and set the element size with a keyword switch:

CREATE/ELEMENTSIZE=*n pathname*

where *n* is the value of the element size.

Multiple Commands

You can enter more than one CLI command on a line by separating the commands with a semi-colon (;). For example:

```
) DIR FUNDS;DEL ACCTC<↓>
```

The commands are executed from left to right, so if any command is invalid, neither it nor anything to the right of it will be executed.

Continuing Command Lines

You can continue a command line to the next line by typing an ampersand (&) before the New Line. The CLI issues the prompt &) on the continuation line. The ampersand, however, is not a delimiter so it must either be preceded or followed by a delimiter if one is required. For example:

```
) DELETE MULTIPLEXORS SYNCHRONOUS &<↓>

&) ASYNCHRONOUS COMMUNICATIONS<↓>
```

There is no limit to the number of continuation lines you can have.

Abbreviations

The CLI allows you to abbreviate all commands and switches. The shortest abbreviation you can use consists of the smallest number of characters, beginning with the first character, which results in a unique specification for a command or switch. So, for example, you can abbreviate **FILESTATUS** to **F** since it is the only CLI command which begins with an F. You cannot, however, abbreviate **DELETE** to **D** because several commands begin with D. Even **DE** would be insufficient because of the command **DEBUG**. In fact, the shortest acceptable abbreviation for **DELETE** is **DEL**.

> **NOTE:** *It is a good idea to spell out command names in macros to save your re-doing macros if any new commands are implemented.*

Parentheses

Parentheses in a command line result in command repetition. If you enter a command followed by an argument list in parentheses, the CLI executes the arguments in the list as if each argument were entered on a separate line. For example, the command line

```
) TYPE (X Y Z)<↓>
```

is equivalent to

```
) TYPE X<↓>

) TYPE Y<↓>

) TYPE Z<↓>
```

If a subset of the entire argument list is in parentheses, the command is repeated for each argument in the subset in conjunction with the remaining argument(s). For example, the command line

```
) COPY (FILEX FILEY) FILEZ<↓>
```

is equivalent to

```
) COPY FILEX FILEZ<↓>

) COPY FILEY FILEZ<↓>
```

If you enter a command line with two or more argument groups in parentheses, the CLI executes the command for the first argument in each group, then for the second, and so on. For example, the command line

```
) COPY (ANIMALS BIRDS REPTILES) &<↓>

&) (CAT DUCK SNAKE)<↓>
```

is equivalent to

```
) COPY ANIMALS CAT<↓>

) COPY BIRDS DUCK<↓>

) COPY REPTILES SNAKE<↓>
```

If you enter a command line with two or more commands in parentheses, followed by an argument, each command in the parentheses is executed with the argument. For example, the command line

```
) (TYPE DELETE) FILEQ<↓>
```

is equivalent to

```
) TYPE FILEQ<↓>

) DELETE FILEQ<↓>
```

Nested Parentheses

When you nest one set of parentheses in another, the nested list is isolated from the outermost list. That is, the nested arguments are treated as a group. For example, the command line

) WRITE (A (B C) D)<↓>

is equivalent to

) WRITE A<↓>

) WRITE B C<↓>

) WRITE D<↓>

If you enter a command line containing parentheses nested to three or more levels, the CLI interprets the outermost parentheses as a repetition operator, the second level as a grouping operator, the third level as a repetition operator, etc. You can nest parentheses to any depth.

Angle Brackets

Angle brackets in a command line result in argument expansion. They can help you code arguments that contain the same character or character combination. The CLI forms arguments by joining each character enclosed within angle brackets with the characters that appear immediately before the left angle bracket and immediately after the right angle bracket.

For example, the command line

) DELETE FILE<P,Q,R><↓>

is equivalent to

) DELETE FILEP FILEQ FILER<↓>

Order of Evaluation

The CLI allows you to nest angle brackets within angle brackets, parentheses within parentheses, angle brackets within parentheses, and parentheses within angle brackets. The CLI first expands the argument list by processing angle brackets from left to right and from inner to outer. When no angle brackets are left in the command line, the CLI processes parentheses from left to right in pairs.

For example, the command line) WRITE (FILE<1,2,3>((B,C)A))<↓>

is equivalent to) WRITE FILE1<↓>

) WRITE FILE2<↓>

) WRITE FILE3<↓>

) WRITE BA<↓>

) WRITE CA<↓>

NOTE: *The* **WRITE** *command is very useful if you want to experiment with angle brackets and parentheses.*

Chapter 5
The Macro Facility

A macro file is a file containing either command sequences or arguments. The purpose of a macro file is to save you time and effort. You can use the name of a macro file either in place of a CLI command or as an argument in a command line. When the CLI sees a macro filename, it substitutes whatever the macro file contains for the macro filename.

For example, if a macro file contains the names of a series of files, and you provide the macro filename as an argument in a command line, the CLI will access each file named in the macro file. Similarly, if a macro file contains a command sequence, and you type the macro filename, the CLI executes the commands in the macro.

Creating and Executing Macros

Macro files can be created in a couple of different ways. For example,

) **CREATE/I** *macroname*.**CLI**<↓>

where **CREATE** is the CLI command for creating a text file; the **I** switch indicates that the input for the macro is from the user console; *macroname* is the name you supply; and the **.CLI** suffix indicates that the file is a macro.

After you type in the command, a double prompt appears. This means that the CLI is accepting lines from the console and writing them into the file. As you write the macro you terminate each input line with a New- line character and end the input with a right parentheses followed by a New-line. You can also type **CTRL D** to terminate your macro file instead of typing a right parenthesis followed by New-line.

You could also use Speed to create a text file which would contain your macro.

When executing macros, you enclose the macro filename in square brackets, i.e., *[macroname]*. The square brackets tell the CLI you are attempting to execute a macro. For an exception to this, see "Implied Macros".

Examples

Suppose there are four files you tend to work on as a group: TURNOVERS, QUICHE, MOUSSE, and FISH. Clearly, you could provide these filenames as arguments every time you used the same command with the files. But you could also put the filenames in a macro file and save yourself some typing.

Using the **CREATE/I** command, the console session would look as follows:

```
) CREATE/I RECIPES.CLI<↓>

) ) TURNOVERS,QUICHE,MOUSSE,FISH<↓>

) ) ) <↓>
```

You could also use Speed to create a text file called RECIPES.CLI. The text file would simply contain a list of the filenames: TURNOVERS, QUICHE, MOUSSE, FISH.

Any time you wished to access this group of files, you would provide the name of the macro as an argument in the command line. For example,

```
) TYPE [RECIPES]<↓>
```

Similarly, suppose there are three commands you always issue at the end of each working session:

```
) COPY FILE.X FILE.1 FILE.2 FILE.3<↓>
```

```
) DELETE FILE.1 FILE.2 FILE.3<↓>

) TYPE FILE.X<↓>
```

The first command copies FILE.1, FILE.2, and FILE.3 into FILE.X. The second deletes FILE.1, FILE.2, and FILE.3. The third displays FILE.X on the screen.

You could create a macro called ENDING as follows:

```
) CREATE/I ENDING.CLI<↓>

) ) COPY FILE.X FILE.1 FILE.2 FILE.3<↓>

) ) DELETE FILE.1 FILE.2 FILE.3<↓>

) ) TYPE FILE.X<↓>

) ) ) <↓>
```

Any time you wished to execute this macro, you would type:

```
) [ENDING]<↓>
```

ENDING.CLI, however, has one obvious limitation since it only executes the commands on files called FILE.1, FILE.2, FILE.3, and FILE.X. If you wanted to write a macro which let you plug in different filenames each time, you could do so by using dummy arguments.

Dummy Arguments

Dummy arguments allow you to create macros without naming specific files, switches, programs, etc. When the macro is executed, the CLI substitutes the actual arguments and switches you supply for the dummy arguments. Thus you could execute the macro with different arguments every time. Dummy arguments and switches must be enclosed by percent (%) signs. There are two types of dummy arguments: single arguments and range arguments.

Single Argument References

Single Argument Format	Description
%*n*%	Expands to argument *n* with all of its switches.
%*n*/%	Expands to all of the switches on argument *n*.
%*n*/*s*%	Expands the *s* switch on argument *n* if the *s* switch is present. If the *s* switch is a keyword switch, the value is included in the expansion.
%*n*/*s*=%	Value of *s* switch if argument *n* has keyword switch *s*.
%*n*\%	Expands to argument *n* without its switches.
%*n**s*%	Expands to all of the switches of argument *n* except the *s* switch.

Table 5.1 Single Argument References

In the case of single argument references, each dummy argument is replaced by a single argument at execution time. In general, the CLI replaces the dummy argument %0% with the macro name and %*n*% with argument *n* when you call the macro. If you enter fewer than *n* arguments at execution time, then any excess dummy arguments are ignored. You can use multiple slashes or back slashes in a single dummy argument, but not both in the same argument.

The CLI replaces each reference to a switch in a macro body by the corresponding switch specified in the macro call. If the switch is not present in the call, the CLI ignores the dummy switch.

Consider the following example: SWITCH.CLI contains the command lines:

```
RENAME %1% TEMP<↓>

RENAME %2% %1%<↓>

RENAME TEMP %2%<↓>
```

The macro accepts two filenames as arguments. When you call the macro with [SWITCH FILEA FILEB], every %1% is replaced by FILEA and every %2% is replaced by FILEB.