SC20-1846-1
File No. S370-22

Systems

APL/CMS User's Manual Programming RPQ MF2608

Program Number 5799-ALK

This publication describes APL/CMS. It also describes the APL/CMS auxiliary processors, which allow the APL program to perform input and output operations to disks, magnetic tapes, line printers, and other devices.

The programming RPQ described in this manual, and all licensed materials available for it, are provided by IBM on a special quotation basis only, under the terms of the License Agreement for IBM Program Products. Your local IBM branch office can advise you regarding the special quotation and ordering procedures.

IBM

Second Edition (March 1975)

This is a major revision of SC20-1846 and makes obsolete that edition. This edition corresponds to Release 2 of APL/CMS and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters.

Changes are periodically made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 and System/370 Bibliography, Order No. GA22-6822, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments concerning the contents of this publication may be addressed to IBM Scientific Center, APL/CMS Publications, 2670 Hanover Street, Palo Alto, California 94304.

This publication provides information on how to use APL/CMS, describes the language and system features unique to APL/CMS, and the auxiliary processors available with APL/CMS.

This manual has the following sections and two appendixes:

- Section 1 describes APL under CMS, the virtual machine concept, logging onto VM/370, related CP, CMS, and APL commands, how APL uses virtual storage, how to save and copy workspaces, and common error situations and what to do about them.
- Section 2 describes the language and system features unique to APL/CMS.
- Section 3 describes CMS facilities for the APL/CMS user.
- Section 4 describes the auxiliary processors that allow APL programs to request certain CMS services:
  - --AP100 the Command Processor that passes commands to CMS and CP.
  - --AP101 the Stack Input Processor that stores and supplies input entries for use by CMS or APL/CMS.
  - --AP110 the CMS Disk I/O Processor that provides sequential and random access to CMS files.
  - --AP111 the FILEDEF I/O Processor that provides sequential access, via QSAM, to any I/O device supported via FILEDEF. This includes readers, punches, printers, and real OS disks (in read/only mode).
- Appendixes that describe translation table options and auxiliary processor return codes.

The APL/CMS system combines the programming features of APL and the virtual machine facility of VM/370. Several subsets of the system can be defined. One can learn a subset of the system, use it to solve problems, and go on to learn more advanced features as the need arises. The study and the use of APL/CMS is as follows:

1. Read Section 1 in this manual. If unfamiliar with APL, one should read the APL Language Manual.
2. Try making use of APL. See Section 2 for language and system features and limits unique to APL/CMS.
3. Read the remainder of this manual. Try using the shared variable facility and the APL/CMS auxiliary processors.

PREREQUISITE PUBLICATIONS

APL Language Manual, Order No. GC26-3847

SUPPLEMENTAL PUBLICATIONS

The following publications also provide information that may be of interest to the APL/CMS user.

APL/CMS Installation Manual, Order No. SC20-1845

APL\360 Primer, Order No. GH20-0689

IBM Virtual Machine Facility/370:

Command Language Guide for General Users, Order No. GC20-1804
System Messages, Order No. GC20-1808
Terminal User's Guide, Order No. GC20-1810

APL\360 User's Manual, Order No. GH20-0906

APL Shared Variables (APLSV) User's Manual. Order No. SH20-9050

CONTENTS

FIGURES

SECTION 1: USING APL UNDER CMS

THE VIRTUAL MACHINE

VM/370 is a system that manages the resources of a single computer so that many different computing systems, called virtual machines, appear to exist. Each virtual machine appears to have a system console, a CPU, storage and input/output devices. It is not necessary to understand the implications of these terms, but you may want to know, for example, that the storage size of the virtual machine affects the size of an APL/CMS workspace.

To use APL/CMS, you need access to a virtual machine. Usually, your VM/370 system operations group can prepare a virtual machine for your use and tell you the name (called the userid) of the machine and a password that controls access to it.

VM/370 maintains a directory of all the virtual machines that share the resources of the real computer. The directory contains the userid, password, accounting information, normal size of virtual memory, maximum size of virtual memory, and a list of I/O devices, of varying properties, identified by numbers. Some of the properties of the machine can be changed during a terminal session (for example, increasing the size of the virtual storage up to the limit specified in the VM/370 directory).

Each user of APL/CMS must have access to a virtual machine. When the user logs on, VM/370 looks up the directory entry and supplies a virtual machine with the appropriate properties. Typically, a virtual machine has a console (for example, an IBM 2741), a CPU, storage, one or more virtual disks, a virtual card reader, punch, and line printer. A virtual disk is a part of the space on a disk device (for example, five cylinders on an IBM 3330).

ESTABLISHING A CONNECTION TO VM/370

Consult with your system operations group for the location and properties of the terminals you can use, and to find out how to establish a connection. The VM/370: Terminal User's Guide describes some of the terminals used for APL. In this manual, an IBM 2741 is assumed. The terminal has two type

elements: one used for VM/370 and one used for APL. Install the VM/370 type element, and establish a connection. The terminal will type a response that will include VM/370 ONLINE or RESTART. If the keyboard is locked, press the attention key. Type LOGON followed by your userid (the name of your virtual machine) followed by M (short for MASK).

```
logon smith m
```

would be used by someone with the user identification SMITH. You may enter lowercase letters, as shown in the example. The machine will usually reply in uppercase letters. Typing the m causes VM/370 to print a mask string to conceal your password. The machine response is:

```
ENTER PASSWORD:
********
```

Enter your password on top of the printed mask string. The machine then prints the time and date, and any information that the operator wants you to know about. If you have problems logging on, contact your system administrator, or check the VM/370: Terminal User's Guide for guidance on what to do.

VM/370 has two major components. The first component is CP, which is the Control Program. It manages the real resources of the installation to provide independent virtual resources, in the form of virtual machines, for its users. CMS (the Conversational Monitor System) is an interactive monitor system that runs under CP and offers user-oriented features which control the virtual machine.

All programs running on a VM/370 system are under control of CP; they may or may not use CMS. APL/CMS is a system that provides the APL language using the facilities of CMS and CP.

LOADING APL/CMS

After your password is accepted, the introductory message is printed, the keyboard is unlocked and CP waits for you to enter a CP command. Now enter the command which causes CP to load the APL system. This is done in one of two ways, depending on how your system operations group has set up the APL system.

The first method is to install the APL type element and type

IPL APL

IPL stands for initial program load. This command initiates the loading of the APL/CMS system. The machine responds:

A P L / C M S
CLEAR WS

When the keyboard unlocks, you can begin to enter APL statements and commands. Instead of the above, you may see:

⌈↑⌈~∈| ⍺*□ ⌊O∈⌈ TO~ ∈⊃ι⌈~

This message is SYSTEM APL DOES NOT EXIST typed with the APL type element. You should verify that APL is the name assigned to APL/CMS at your installation.

The second method of invoking APL/CMS is to enter

ipl cms
apl

You will then be asked to install the APL type element and press the return key. The system responds as described in the previous paragraph. You can now use APL. You can log off the system by typing the APL command)OFF. The following is an example of a short session at the terminal.

```
d'x38z irvyr;      vm/370 online

l smith m
ENTER PASSWORD:
********
LOGON AT 18:01:18 PDT TUESDAY 03/19/74

IPL APL

A P L / C M S
CLEAR WS
      2+2
4
      2×3
6
      3÷2
1.5
     )OFF
CONNECT= 00:08:45 VIRTCPU= 000:00.66 TOTCPU= 00C:02.20
LOGOFF AT 18:10:03 PDT TUESDAY 03/19/74
```

CP, CMS, AND APL COMMANDS

Some of the messages that you enter at the terminal will be treated as commands. The kind of command that can be used at any time depends on the current environment. There are three environments: CP, CMS, and APL. When you first log on, you are in a CP environment. If you IPL CMS, then you are in the CMS environment. If you IPL APL (or enter 'apl' when in the CMS environment), then you go to the APL environment. A summary of the way the environment changes is:

To go from	To	Enter
Not logged on	CP	logon
CP	APL*	ipl apl
CP	CMS*	ipl cms
CMS	APL*	ipl apl
CMS	APL	apl
APL	Logged off	)OFF
APL	CMS	)OFF HOLD†
APL	CP	)OFF HOLD†
CMS	Logged off	logoff
CP	Logged off	logoff
CMS	CP	CP

---

*Note that IPL APL and IPL CMS are not available in some installations.

†You return to CP if you invoked APL by an IPL.

When you are in the CP environment, all input lines are treated as CP commands. When you are in the CMS environment, input lines are treated as CMS commands; if they are not recognized as CMS commands, then they are treated as CP commands. When in the APL environment, all input lines are treated as APL statements or commands. CMS does not accept APL commands, nor does APL obey CP or CMS commands.

If you have made a typing error while entering a CP or CMS command, then you have a chance to correct the error if you have not yet pressed the return key. Entering ¢ causes the machine to ignore everything to the left of the ¢. Entering @ causes the previous character to be ignored; entering @@ causes the two previous characters to be ignored, and so on. For example,

```
ipx a¢ ipl cms
iq@pl cms
ipl dm@@cms
```

all have the same effect as if "ipl cms" had been typed without any errors. If you make an error and fail to correct it, then you must wait until CP or CMS has processed it; typically, the result is an error message indicating an unknown command, the keyboard unlocks, and you can try again. (The @ and ¢ are the characters ¯ and ≥ of the 987 APL type element and ← and → of the 988 APL type element.) These VM/370 logical line edit characters can not be used while in the APL environment.

When a CMS command has finished execution, it responds as follows:

```
R; T=n.nn/x.xx  hh.mm.ss
```

This is called a Ready message. n.nn and x.xx are the virtual and real CPU times (in seconds and hundredths of seconds) used since the last ready message. hh.mm.ss is the time of day (hours, minutes, and seconds). If an error has occurred, then the R is replaced by R(nnnnn) where nnnnn is an error code; in most cases an explanatory message appears on a previous line.

USE OF VIRTUAL STORAGE

APL/CMS uses an area of storage called a workspace. The workspace contains user defined functions and data, some space for system tables, and a work area. As new functions and data are defined, the space they need is taken from the work area. The amount of work area can be determined by the APL system variable □*WA*. In some APL systems, the size of the workspace is fixed when the system is generated and all users have the same workspace size. In APL/CMS the size of the workspace is determined dynamically when you load APL. After space is allocated for the system and for input/output areas, the rest of virtual storage is allocated to the workspace. Subsequently, when loading a workspace, its work area is adjusted accordingly.

The space used by APL/CMS varies from installation to installation, but a typical figure is about 340,000 bytes, not including the workspace. The size of your virtual storage is determined by an entry in your VM/370 directory. When in the CP or CMS environment, you can find out what the size is by entering the command:

```
query storage
```

The reply may be in units of K (for kilo or 1024) bytes or M (for mega or 1,048,576) bytes. If the reply is, for example, 512K then the work area in the workspace is approximately 170,000 bytes. As you become familiar with APL, you should be able to relate the workspace size to the kinds of APL programs you can run. The maximum virtual storage allowed by VM/370 is 16M bytes. The maximum work area under APL/CMS is about 16,400,000 bytes.

SAVING AND RESTORING WORKSPACES

CMS refers to virtual disks by one of the letters A through G, S, Y, and Z. The distributed version of APL/CMS uses the following disks:

A -- User's private library
D -- Temporary space (used during)COPY, for example)
G -- Libraries 1000 through 999999 (read/write access)
Z -- Libraries 1 through 999 (read-only access)

The A-disk and D-disk belong to your virtual machine, and normally only your machine has access to them. The other disks belong to another virtual machine which every APL/CMS user can access. The APL)SAVE command stores a workspace on the A-disk. A library number causes the public library to be accessed. A workspace cannot be saved in a read-only library. The read-only library is usually maintained by the APL system librarian. Everyone may save a workspace in the read/write public libraries if they exist.

In some APL systems, you can load a workspace from another user's private library by specifying that user's account number (and password if necessary). Under APL/CMS, each private library is maintained on the user's A-disk. Users cannot directly access another user's private library. Numerous methods exist for transferring workspaces between users in a secure manner. See "Sending Workspaces to Other APL/CMS Users" in Section 3.

APL/CMS sets no limit on the number of workspaces that can be stored in the private library; the limit depends upon the total amount of disk space available. If insufficient space is available to perform a)SAVE, then a DISK FULL error occurs. You can make room with the)DROP command. Section 3 contains an explanation as to how you can find the amount of disk space used by each workspace.

If the read/write public libraries (G disk) are full, consult the APL/CMS systems staff at your installation. If the message DISK NOT AVAILABLE appears when attempting to save a workspace in the public libraries, try it again as another user was probably accessing that disk momentarily.

ERRORS

Most of the error messages are from APL and are described in the APL Language Manual. You may also get the error messages discussed in this section.

NOT IN CP DIRECTORY

This message will be received when attempting to logon when you are not an authorized user. It may also appear if you made a typing error, or if there was line noise creating a problem.

PASSWORD INCORRECT

See the installation manager to find out your correct password to logon. Unlike some APL systems, you cannot directly respecify this password. Typing errors or line noise may cause this message to appear.

DMSACC112S DISK 'A(191)' DEVICE ERROR

This message occurs when your disk is not in the proper CMS format and must be formatted using the CMS FORMAT command. Unless this is done, APL/CMS does not have the capability to access your private library.

SYSTEM APL DOES NOT EXIST

This message is an indication that your installation has generated APL/CMS under a different name or that APL/CMS is available only from the CMS environment.

CP (keyboard unlocks)

The letters CP are typed as the result of telephone-line noise or importunate use of the attention key. Type the word BEGIN (B for short) and press the return key. If you are attempting to generate a double attention, type EXT and press the return key.

COMMAND COMPLETE ... CLEAR WS

This message (ellipsis indicates several lines of print) may be caused by an error in APL/CMS. Report the occurrence(s) to the system operations group. You can continue to work with APL, but your active workspace has been replaced by a clear workspace.

CP ENTERED, REQUEST PLEASE.

This message indicates that you are in the CP environment and must reload APL/CMS.

RESTART

-- or --

VM/370 ONLINE

Either message indicates that VM/370 has been restarted. You must logon to the system and reload APL/CMS. Your active workspace has been lost.

communication line loss

You may lose the active workspace because of an abnormal disconnect of the telephone connection to VM/370. A 15-minute time period is provided to allow you to reestablish the communication line and logon again. For the method of resuming APL execution refer to Section 2 under the heading "The CONTINUE Workspace".

If you have to execute a long running program you may wish to run with the terminal disconnected. Load the workspace 1 APFNS and type DESCRIBE for additional information and a function to disconnect the terminal.

SECTION 2: LANGUAGE AND SYSTEM CONSIDERATIONS

This section discribes the language and system features unique to APL/CMS. It also gives various system and language limitations due to the implementation. These are grouped in terms of keyboard I/O operations, error handling, function definition, system commands, system variables, and shared variables.

KEYBOARD ENTRY AND OUTPUT

DOUBLE ATTENTION (STRONG INTERRUPT)

Generating a double attention (strong interrupt) under APL/CMS requires a fine touch as CP monitors attentions for possible entry to that environment. Pressing the attention key twice with deliberate speed should suffice. If the characters)* are typed, indicating entry to the CP environment, simply type EXT; the effect is the same as a properly executed double attention. (If you have entered the CP environment and wish to generate a single attention (weak interrupt), not a double attention, type BEGIN.)

INPUT LINE LIMITATION

The maximum number of keystrokes which can be entered, excluding the terminal carriage return, is 130. If the entry is interrupted by the attention key, causing a caret to appear, then the keystroke count is begun again. Since input of this form appends to the line already entered, it is possible to enter more than 130 keystrokes by striking the attention on every group of 130 or less, the final group being terminated by the carriage return. The system limitation for input of this form is 762 keystrokes, including one character for each attention and carriage return. These input keystrokes must produce less than 380 APL characters.

OPEN QUOTE

All keyboard entries are terminated by a carriage return. A keyboard entry containing an open quote will invoke no special system treatment. It will produce a SYNTAX error report, similar to an entry with unbalanced parentheses.

CHARACTER ERRORS

The carriage return (except to terminate a keyboard entry) and the horizontal tabulate are invalid input characters. The terminal control characters backspace, horizontal tabulate, idle, linefeed, and new line are available as a five element character vector, QUADTC, in the workspace 1 SPECIAL.

COMMANDS DURING FUNCTION DEFINITION

During function definition, every keyboard entry beginning with a left bracket (supplied by the system for convenience) is treated as an EDIT or INPUT request. APL/CMS reports some errors (for example, SYNTAX errors) on input but the entry is accepted. A keyboard entry that does not begin with a left bracket is treated as an immediate execution entry. To produce an entry of this form, backspace and use the attention key to erase the beginning of the line provided by the system.

The following statements apply to immediate execution when in function definition mode:

- If a workspace is saved during function definition, then a subsequent loading of the workspace resumes the function definition at the point when the save command was given.
- If a function, saved during edit mode, is copied from a workspace, it will be copied in closed form.
- Any attempt to enter or leave function definition results in the message DEFN ERROR.
- Any action explicitly or implicitly causing an erasure of the function being edited terminates function definition mode.

ERROR HANDLING

LOCKED FUNCTIONS

Execution cannot be suspended within a locked function. However, a pendent locked function can occur if execution is suspended within an unlocked function used by that function.

STACK FULL

A portion of workspace storage, called the stack, is used to hold APL expressions during the execution of an APL statement, and to hold status information during the execution of a user-defined function. If the space allocated to the stack has been used, a STACK FULL error message is typed. For corrective action, use)SI to display the state indicator. Clear the stack by repeated use of the '→' key. To change the size of this area, use the)STACK command as described in the Section 2 under the heading "System Commands"

STACK DAMAGE

The error report SI DAMAGE ENCOUNTERED is issued when execution can proceed no further because of damage to the execution stack. This may be caused by erasure of a pendent function, for example.

FUNCTION DEFINITION

IMMEDIATE EXECUTION

Immediate execution of most APL statements can be effected while in function definition. See this section under the heading 'Commands During Function Definition' for details.

LINE DELETION

A line of a function is deleted by entering a left bracket, a delta, the line number and a right bracket. For example:

```
[99]   [Δ4]
[4]    [Δ5]
[5]
```

will delete statements 4 and 5. In the above example the [99], [4], and [5] were supplied by APL and the other characters were entered by the user. Multiple lines cannot be deleted with a single edit request. The attention signal cannot be used to delete a line.

LINE DISPLAY

Lines of a function which have been modified or added are not put into canonical form until the function definition is closed. For example, extra spaces are not removed until the function definition is closed.

FUNCTION HEADER

A function header which contains more than one occurance of the same name is rejected and the report DEFN ERROR is given.

LABELS

Names used as labels within a function are active only when that function is being executed or is the top-most suspended function and during those times, labels act as local constants. At other times, when the function is pendent, and within any other defined function, these names are unencumbered by their use as labels.

SYSTEM COMMANDS

COMMUNICATION COMMANDS

In order to send messages to other virtual machine users, the commands)PORTS,)MSG and)OPR are supplied. The command)PORTS produces a list of userids of all active virtual machines. The command)MSG followed by a userid followed by text sends the text to that userid. The userid should be delineated by blanks. The command)OPR followed by text sends the text to the VM/370 system operator.

There are occasions when a user may wish to be undisturbed by messages arriving from another terminal. The command)MSG OFF blocks all messages from other terminals and the command)MSG ON restores the acceptance of messages.

THE COPY COMMAND

This command can be used to copy a workspace, a group, a function, or a variable. Only one of these objects can be copied at a time. The workspace 1 SPECIAL contains a function called COPY which will cause more than one object to be copied. (Type DESCRIBE after loading that workspace for information.)

THE LIB COMMAND

The output of the)LIB command under APL/CMS is an unordered list of workspace names in that library. No provision is made for printing a subset of that list.

THE STACK COMMAND

A portion of workspace storage, called the stack, is used to hold APL expressions during the execution of an APL statement, and to hold status indication during the execution of a user defined function. The amount of space allocated to the stack may be displayed by)STACK and changed in a clear workspace by the command)STACK N. When a clear workspace is loaded, N has the value 512.

ENVIRONMENT COMMANDS

)DIGITS N,)ORIGIN N, and)WIDTH N can be used to set the global values of ⎕*PP*, ⎕*IO*, and ⎕*PW*. If N is outside the range of meaningful values for the system variable then INCORRECT COMMAND is reported and the value of the system variable is unchanged. If the previous value of the system value is within the meaningful range, that value is printed in the message WAS N. See "System Variables" in this section for the meaningful range of these variables.

WORKSPACE IDENTIFICATION

APL/CMS supports a maximum of eight characters as the workspace identification. If more than eight characters are used to save, load, or copy a workspace, a WS NAME IS TOO LONG report is given.

THE CONTINUE WORKSPACE

If a workspace named CONTINUE exists in the user's private library and does not have a lock, it is loaded when APL/CMS is invoked.

If the machine loses a connection with a terminal, then the active workspace is not saved. However, VM/370 provides a grace period of about 15 minutes for the user to establish a new connection and logon on to the system. Once reconnected, type:

```
term apl on attn off linesize 130 mode vm
begin
```

to resume APL execution.

APL/CMS can be run normally in a disconnected state for long jobs which do not require a terminal. A function in 1 APFNS provides this capability. The example shown in Section 4 for auxiliary processor 101 demonstrates a way to checkpoint a workspace by use of an APL function.

SYSTEM VARIABLES

See the APL Language Manual for a description of system variables. The following information gives system dependent values.

RANGE OF VALUES

Name	Meaningful Range	Value in Clear WS
□*CT*	0 *TO* 2*¯32	1*E*¯13
□*IO*	0 *AND* 1	1
□*PP*	1 *THROUGH* 16	10
□*PW*	30 *THROUGH* 254	120
□*RL*	1 *THROUGH* ¯1+2*31	7*5

(The meaningful values are integers except for □*CT*.)

In addition:

□*AI* The accounting information is user identification (always zero), virtual computer time, connect time, keying time in milliseconds, presented as a four element vector.

□*AV* The atomic vector is dependent upon internal character codes of APL/CMS. See below.

□*TS* The time stamp is year, month, day, hour, minute, second, millisecond.

□*TT* The terminal type is 0 for 1050, 1 for Selectric, 2 for PTTC/BCD, ¯1 for other devices.

□*UL* The user load is always reported as 1.

□*WA* The working area is given in bytes.

Atomic Vector

The workspace 1 SPECIAL contains three permutations of ⎕*AV* named QUADAVAPLSV, QUADAVBINARY and QUADAVEBCDIC. QUADAVBINARY is arranged so that the internal character codes are in ascending binary sequence starting with zero. QUADAVAPLSV is arranged so that the APL characters are at the same index positions as in ⎕*AV* in the APLSV system. QUADAVEBCDIC can be indexed by the numeric value of EBCDIC codes to select characters common to APL and EBCDIC. Further details are contained in the workspace 1 SPECIAL.

Users with a requirement for the terminal control characters backspace, horizontal tabulate, idle, line feed, and new line can copy 1 SPECIAL QUADTC to obtain these as a five-element character vector.

SHARED VARIABLES

Shared variables are used to communicate between independent processors. The kinds of processors which can be used depend on the facilities of the host operating system. APL/CMS has a single APL user on each virtual machine, running under the CMS component of VM. Processors available in APL/CMS include the single user's APL program, the APL/CMS processor and four APs (auxiliary processors) which are supplied with the system. The APs are described in detail in Section 4; for example, AP110 can read and write CMS disk files. APL/CMS does not allow active APL users to communicate directly with each other through the use of shared variables.

OFFERS

If a user attempts to share more variables than the limit determined when the system is installed the error reported is INTERFACE QUOTA EXHAUSTED. This limit is 45 in APL/CMS as distributed. Other system limits may cause the report SVP MEMORY FULL or SVP NAME TABLE FULL to be issued; these limits are described in the APL/CMS Installation Manual and may be changed when the system is installed.

USING THE APL/CMS AUXILIARY PROCESSORS

The APL/CMS auxiliary processors use the initial value of the shared variable to indicate the source and/or destination of the processors input/output and to select certain options. For example, the following sequence will write two records to a CMS file called "EXAMPLE SCRIPT". The "370" conversion option is selected. The first record is "THIS GOES IN THE FIRST RECORD" and the second record is "AND THIS GOES IN THE SECOND".

```
      S←'EXAMPLE SCRIPT(370'
      110 ⎕SVO'S'
2
      S←'THIS GOES IN THE FIRST RECORD'
      S←'AND THIS GOES IN THE SECOND'
      ⎕SVR'S'
2
```

In this case, S is the name of the shared variable. Notice that the argument to ⎕SVO and ⎕SVR is 'S' not S. See Section 4 for further details of the auxiliary processors.

COMPATIBILITY

APL/CMS has one user on one virtual machine. In addition, CMS is a synchronous system, that is, one which completes each request before continuing with the next. Some APL systems have many users on one real machine and it is possible for asynchronous processes to communicate with each other. APL/CMS provides certain features which have little value in the CMS environment but which may be useful for maintaining compatibility with these other APL systems.

Surrogate Names

APL/CMS allows the use of surrogate names, but since its auxiliary processors will accept any names offered to them, surrogate names need not be used.

Access Control

The APL/CMS auxiliary processors set the access control vector to 1 1 1 1, the highest degree of control possible.

If a valid offer is made to a nonexistent processor then the access control vector is set to zero, for example

```
      400 ⎕SVO'X'
1
      ⎕SVC'X'
0 0 0 0
      X←99
      X
99
      1 1 1 1 ⎕SVC'X'
1 1 1 1
```

The access control vector now specifies that successive accesses by the APL program requires an intervening access by processor 400. Processor 400 does not exist so it will neither use or set X. If the APL program attempts to access X then a permanent wait would result. APL/CMS detects this situation, forces an interrupt, and prints the error report INTERRUPT: PERMANENT SV WAIT.

DISTRIBUTED WORKSPACES

APL/CMS has certain standard workspaces which are distributed with the system. The names of all the workspaces can be displayed by use of the APL command)LIB 1. The purpose and method of using each workspace can be displayed by loading the workspace and typing DESCRIBE. A short description of each workspace follows.

APFNS

Contains functions to simplify and facilitate the use of the APL/CMS auxiliary processors, especially AP111.

APLCOURS

Provides computer assisted instruction in the use of APL functions.

ADVANCED

Provides examples of the APL programs which the user can study and trace to further an understanding of APL.

CETEST

Contains functions to test typewriter terminals.

CONVERT

Provides aid in converting APL\360 and APLSV workspace dump tapes.

FORMAT

Illustrates the use of dyadic format and the scan operator.

NEWS

Describes differences between APL/CMS and APL\360.

PLOTFORM

Contains programs to graph functions on the typewriter using the APL type ball.

PRINT

Contains programs for transmitting APL programs and data to a line printer equipped with an APL print train.

REL2NEWS

Describes differences between release 1 and release 2 of APL/CMS.

SPECIAL

Contains a character vector whose elements are terminal control characters and three useful permutations of ⎕*AV*.

TYPEDRIL

Provides computer assistance in improving typing skills.

WSFNS

Provides certain functions, such as ORIGIN, which were used in APL\360 programs.

XREF

Contains APL programs which can be used to list APL programs, to prepare cross reference information on functions and variables, and to display topological relationships among programs in a workspace.

SECTION 3: CMS FACILITIES AND THE APL/CMS USER

In this part of the manual, we discuss some CMS concepts and a subset of CP and CMS commands that may be useful to the APL user. For further details, see the VM/370: Command Language Guide for General Users. CMS commands cannot be used in the APL environment directly but can be transmitted via auxiliary processor 100 described in Section 4. APL commands are not recognized by CP or CMS.

LINK AND ACCESS COMMANDS

The VM/370 directory contains the properties of individual virtual machines. One or more entries specify the virtual disk(s) that belong to a particular machine. CP identifies disks by a device address. Most machines have a primary disk at device address 191 and will have links to disks belonging to other machines. For example, the directory for an APL user may contain an entry of the form:

```
link vmaplsys 101 101 rr
```

This indicates the user has access to the 101 disk of a machine whose userid is VMAPLSYS. The last two operands define the device address as 101 and the access to the disk as read-only for this user.

As noted in the Section 1 discussion on "Saving and Restoring Workspaces", CMS refers to disks by mode letter. A correspondence between the CP device addresses and the CMS usage of the disks is established by an ACCESS command. For example,

```
access 101 z
```

allows CMS to use disk 101 as a Z disk. CMS accesses 191 as the A-disk and APL/CMS accesses 101 as the Z disk.

The CP command LINK and the CMS command ACCESS, can be used, among cooperating users, to establish a common disk for storage of workspaces and data files. Access to a user's private disk is controlled by VM/370 directory entries and passwords.

CMS FILES

Items on a virtual disk are organized into CMS files. A CMS file is known by its filename, filetype, and filemode. The filename and the filetype are each composed of one- to eight-alphameric characters. The filetype is formed like a filename, but CMS commands usually associate a particular filetype with a particular kind of file. For example, APL/CMS uses the filetype VMAPLWS for all workspaces in the private library. Files with EXEC filetype usually contain a list of CMS and CP commands to be executed. The filemode is one letter followed by one number. The letter specifies the disk the file is on. The numeral 1 indicates a permanent file available for both reading and writing.

The information in CMS files is grouped into records, which are the smallest unit in the file system. Records are grouped into files. Records in each file are fixed length, meaning the length of all records is the same (and must remain so) or variable length, which imposes no such restriction. The maximum length of a record is 65,532 bytes. CMS allows its files to be accessed sequentially or by specification of the record position (1-origin).

Random accessing can be used with files of fixed or variable length records but it is more efficient with fixed files. Records can be changed in existing files, but the length of any record of a variable file, except the last record, cannot be changed without unpredictable results. Records cannot be deleted from files but can be added to the end of an existing file.

The units of information which CMS transmits to and from disk storage are called blocks. The CMS file system collects several short records or splits up long records as it transfers information block by block to and from disk. A block is 800 bytes long. You need be aware of blocks only if you wish to compute the amount of disk space used.

LISTFILE COMMAND

At some stage of APL execution, you may encounter a DISK FULL message. You can use the APL)LIB command to list the workspaces on the disk; however, you may have other files generated by the APL auxilary processors which are not listed by)LIB. You can list all files with the CMS LISTFILE command which has some of the following variants:

```
listfile fn ft fm
```

displays information concerning the file having the given filename (fn), filetype (ft), and filemode (fm). If fm is omitted then A1 is assumed; if ft and fm are omitted then all files with name fn on the A-disk are listed.

```
listfile * ft fm
```

lists all files with type ft on disk fm. The command

```
listfile * vmaplws
```

lists all VMAPLWS files, and

```
listfile
```

lists all files on the A-disk. There are several options that can also be used. If the command line ends in (ALLOC), then the format (fixed or variable), the number of records, and the number of blocks are listed for each file. For example:

```
listfile alpha (alloc)
FILENAME FILETYPE  FM  FORMAT    RECS BLOCK
ALPHA    VMAPLWS   A1  V  4616     3     9
ALPHA    ASSEMBLE  A1  F    80   392    40
```

"V 4616" denotes variable length records, with the longest record of length 4616 bytes. "F 80" denotes fixed-length records of 80 bytes each. The number under BLOCK gives the number of 800-byte blocks occupied by the file.

QUERY COMMAND

The CMS command LISTFILE lets you find the space used on a virtual disk by each file. More generally, the CMS command

```
query disk a
```

gives a summary of the state of the A-disk (other disks may be specified by their letter). An example of the reply to a query is,

```
A (191): 3 FILES; 155 REC IN USE, 1173 LEFT (of 1328),
         12% FULL (5 CYL), 3330, R/W
```

Do not be confused by this inconsistent usage: REC refers to what LISTFILE calls blocks. The reply shows that 12 percent of the available space is used. The virtual disk address is 191 and is 5 cylinders of 3330 disk storage, which is linked in read/write mode.

As mentioned earlier, the CP command QUERY STORAGE, indicates to you the size of the user's virtual storage. The CP command QUERY NAMES lists the names of the other virtual machines currently logged on the system.

You can send messages by using the APL)MSG command, or the CP MSG (or MESSAGE) command "cp msg userid ...any message... " where userid is the name of the other virtual machine. (This use of the userid corresponds to the APL\360 use of the port number.)

ERASE COMMAND

You can free the disk space used by unwanted workspaces by issuing the APL)DROP command. You can get rid of workspaces or other unwanted CMS files by using the CMS command

```
erase fn ft fm
```

If fm is omitted, then A is assumed. Replacing fn with an asterisk (*) causes all files of type ft to be erased.

WORKSPACES AND CMS FILES

An APL)SAVE command causes the APL/CMS system to store the information from the workspace into a CMS file. Only useful information is stored, not the work area, so the space on the disk is related to the APL objects in the workspace, not the gross workspace size. The A-disk is used for storage of these workspaces when no library is selected. Libraries are stored on disks apart from the APL user's virtual machine; details are supplied in the APL/CMS Installation Manual.

APL/CMS uses the D-disk, if accessed read-write, otherwise the A-disk, for storing intermediate and utility files of type VMAPLUT. These files are used for the APL stack of auxiliary processor 101, the workspace conversion utilities, and during the APL)COPY and)PCOPY commands, for example.

PRINT, PUNCH, AND TYPE COMMANDS

An APL program can create files by using the auxiliary processors (see Section 4). One of the processors can send files directly to the line printer or card punch. There are some occasions when it is easier to use the auxiliary processor to create a CMS file and then use CMS commands to type or print the file. When the CMS PRINT command is issued, as follows:

```
print fn ft fm
```

it causes the contents of the specified CMS file to be printed. The graphical representation of each character is dependent upon the printer characteristics. The VMAPLWS files contain nonprintable records; however, the auxiliary processors supplied with the system can be used to write out APL variables in printable format.

The PUNCH command operates like the PRINT command except that eighty characters of the records are punched into cards. The punch reproduces any character. Cards may be read into the system with the CMS READCARD command. For further details, see VM/370: Command Language Guide for General Users.

The TYPE command is similar to the PRINT command except that output appears at the terminal. If the file was produced with the APL conversion option of the auxiliary processor AP110, then the commands:

```
cp term apl on
TYPE FN FT FM
CP TERM APL OFF
```

should be used to type the file. The first command indicates that you are going to use your APL type element for typing; the third command indicates the subsequent use of the VM/370 type element.

EDIT AND SCRIPT COMMANDS

CMS has an editor, called EDIT. For additional information about the editor, see the publication VM/370: EDIT Guide. The SCRIPT processor, an Installed User Program (IUP) available through IBM for a license fee, can be used to prepare formatted and paginated output, like this document, from CMS files.

FILEDEF COMMAND

This CMS command is used to simulate the functions of the OS Job Control Language Data Definition (DD) in the CMS environment. Device independence is achieved by allowing the unit specification and file characteristics to be transmitted to programs that use the OS simulation macros and functions. APL/CMS includes an auxiliary processor, AP111, which supplies sequential access via QSAM to any device specified by use of the FILEDEF command.

Note: The CMS file processor, AP110, does not use FILEDEF and does not need any DD information. The FILDEF commands most commonly required by AP111 are issued automatically by some functions in the 1 APFNS workspace. (Type DESCRIBE after loading that workspace.)

COPYFILE AND MOVEFILE COMMANDS

The COPYFILE command can be used to copy part or all of a CMS file, to combine files, to change record formats and to do various transformations on data in the file. The MOVEFILE command moves data from any device supported by VM/370 to any other device supported by VM/370. The input and output devices are defined by use of the FILEDEF command.

TIME

When you log off the system, a time message is issued as follows:

```
CONNECT hh:mm:ss VIRTCPU mmm:ss:hs  TOTCPU mmm:ss:hs
```

where:

hh	is hours
mm or mmm	is minutes
ss	is seconds
hs	is hundredths of a second

The connect time is the elapsed time since you logged on. The VIRTCPU is the virtual CPU time you have used. TOTCPU is VIRTCPU plus the CPU time that was spent in CP.

The CP command QUERY TIME yields the same result.

SAVING WORKSPACES ON MAGNETIC TAPE

TAPE Command

The TAPE command can be used to save and restore CMS files on magnetic tape. It is useful for saving infrequently used workspaces or sending workspaces to other APL/CMS system installations. To save one or more files on a tape:

1. Log on to your machine.

2. IPL CMS

3. Ask the operator to ready the tape and attach it as 181 to your machine by issuing the following message:

   ```
   cp msg op pls attach tape pasc123 as 181 with ring in
   ```

   (This sample message is based on the assumption that the tape has "PASC123" as a label.) The phrase "ring in" tells him to put a file protect ring in the tape; if the ring is out then it is impossible to write on the tape. In general, the operator will notify you when the tape is ready.

4. To dump the workspaces ALPHA and BETA, issue the following commands:

   ```
   tape rew
   tape dump alpha vmaplws
   tape dump beta  vmaplws
   tape wtm
   tape rew
   tape scan
   ```

Note that each command starts with the word TAPE followed by a space. REW causes the tape to be rewound, but note that it must be spelled REW, not REWIND. TAPE DUMP writes the named CMS file onto the tape; you may repeat this command for as many files as you wish to dump. The TAPE WTM command writes a tape mark on the tape. A tape mark is conveniently used to separate groups of workspaces. If you wish to dump all workspaces, then you can use:

```
tape dump * vmaplws
```

TAPE SCAN Command

The TAPE SCAN command reads the tape, verifies the tape contents, and types out a list of the CMS files on the tape from its current position to a tape mark.

If the tape appears to be satisfactory then use the following CP commands:

```
cp detach 181
cp msg op remove ring, and save tape
```

to unload and save the tape.

To retrieve a workspace at a later date, then log on, IPL CMS, and issue the command:

```
cp msg op please attach pasc123 as 181 with ring out
```

When the tape is attached, you can then issue the commands:

```
tape rew
tape load beta vmaplws
```

to transfer the CMS file BETA VMAPLWS to your A-disk, which will overwrite an existing file of the same name. The example shows just one file being loaded. Using the command TAPE LOAD with no additional arguments loads all the workspaces up to a tape mark. (You should detach unit 181 when you no longer need it.)

If there is insufficient space on your disk, then you may get a DISK FULL message. You can, if you wish, make some space available on the disk, rewind the tape, and try again.

SPOOL Command

This CP command has many options to control the disposition of files associated with your virtual card reader, card punch, and line printer. For example:

```
spool printer copy n
```

specifies that n copies be made of your line printer output. The command:

```
spool punch to userid
```

causes your virtual punch output to be directed to the virtual card reader of the machine identified by userid.

```
spool punch off
```

directs future output to the real card punch.

DISK DUMP and DISK LOAD Commands

The CMS command, DISK, can be used to move files from disk to card format using the DUMP option, or the reverse using the LOAD option. It is generally not used to punch real cards (the TAPE command is normally used to move files from the system); rather, it is used in combination with the SPOOL command to direct files to, or receive files from, another user. For example, the commands

```
cp spool punch to johndoe
disk dump alpha vmaplws
cp spool punch off
```

transfer the APL/CMS workspace called ALPHA to the card reader of the machine, JOHNDOE. At that machine, the command

```
disk load
```

creates the file ALPHA VMAPLWS on the A-disk.

Note: Use the SPOOL command before issuing the DISK DUMP command. If the output is not spooled to another user, it is punched on real cards. If you discover you have made such a mistake, immediately send the following message:

```
cp msg operator  please flush punch output
```

USING APL\360 OR APLSV WORKSPACES

To transfer APL\360 workspaces to an APL/CMS system requires several steps. Get the APL\360 system staff to dump your workspaces onto magnetic tape. Workspaces from an APLSV system must be dumped in APL\360 compatible form using the level 0 option of the APLSV utility program. This tape is not in the format of the CMS dump tape, so your system staff must convert the tape to an APL/CMS workspace on your disk.

If you are familiar with CMS, then you may wish to do the conversion yourself. The process is described in the APL/CMS Installation Manual.

The conversion goes in two steps. Suppose the tape contains a workspace from APL\360 called ALPHA. The first step reads the tape and produces a CMS file with the name of ALPHA and filetype of APLWS. The second step accepts ALPHA APLWS as input, converts this workspace to the form used by APL/CMS and produces the file called ALPHA VMAPLWS. The same conversion procedure can be used for APLSV workspaces. APL(CMS) IUP workspaces already exist as APLWS files, have the same format as APL\360, and require only the second conversion step.

After conversion, the workspace may contain functions with the names BADHEADERn, where n goes from 1 to the number of such names. The original header of these functions contains at least one duplicate name and is stored as a comment in the first line of the function. Edit the function, correct the original header, delete the lamp and del symbols, change the line number to zero, and close the function.

The transfer of workspace information to APLSV (or other foreign APL systems with I/O capability) can be effected by writing the canonical representation of the functions in the workspace along with the variables in the workspace onto magnetic tape. Read this tape with APL programs to create a workspace on APLSV.

SENDING WORKSPACES TO OTHER APL/CMS USERS

The magnetic tape produced by the TAPE DUMP command can be used to send workspaces to other APL/CMS users. If the other user is on the same physical machine as you, then it is more efficient to do a direct disk-to-disk transfer using the APL/CMS public library. Although an installation option, you can normally)LOAD and)SAVE into libraries 1000

through 999999. Loading the workspace you wish to transfer and saving it in the public library makes it available to another user, who can drop it from the public library after the transfer.

If you get the message

DISK NOT AVAILABLE

while attempting the)SAVE in the public library, it means that the disk is temporarily unavailable for writing because some other machine is writing on it. Retry the command in a few seconds. There is one drawback to this method, namely the lack of privacy during the transfer. A more secure transfer is via the previously described DISK command.

Once a workspace has been transferred, it can be used by entering the APL environment and using the)LOAD command. The load command adjusts the size of the workspace to the available storage. If the virtual storage on your machine is too small to accommodate the workspace, the message

WS TOO LARGE

results and loading does not take place.

If you need to accommodate a workspace larger than your virtual storage, you may return to the CP environment and issue a DEFINE STORAGE command to expand the size of your virtual storage (within limits set by your installation manager). Reload APL/CMS and try again.

THE CMS BATCH FACILITY AND APL/CMS

VM/370 has a facility which allows CMS jobs to be run in batch mode. This facility may prove useful if you have an APL program which runs for a long time. This job can be sent to another virtual machine (the CMSBATCH machine) and it will be run on that machine; meanwhile, you can use your machine for any other job, such as interactive APL/CMS.

To use the batch facility you must first of all prepare a CMS file. The file may have any name, but must have file type MEMO. Each line of the file should contain one line of APL input. Typically, the line will be an APL command or an APL statement for immediate execution. The file can be prepared using the CMS editor, or alternatively it could be written from an APL workspace using auxiliary processor 110.

When the file has been prepared, it should be sent to the CMSBATCH machine by using the CMS command BATCHAPL. The BATCHAPL command takes five arguments which are: 1) the userid to be used in charging for time on the batch machine, 2) the account number, 3) an arbitrary name used to identify the job, 4) the name of the MEMO file, 5) an optional argument which is described below. As an example, BATCHAPL SMITH 999 TEST1 INPUT1 will send a job which is to be called TEST1 to the BATCHCMS machine. The job is sent as a series of card images which go into the card reader of the batch machine. The first card specifies the accounting information. Subsequent cards cause the APL/CMS system to be loaded in the batch machine. APL/CMS operates in the normal way except that it receives input from the virtual card reader; each line of TEST1 MEMO is read as one line of APL input, and it generates the normal APL response.

Since the batch job is being run on another machine, it has access to the APL public libraries, but it does not have access to the user's private library. If the job requires a workspace from the private library then this workspace must be sent to CMSBATCH by specifying its name as the fifth argument of the BATCHAPL command. As an example, suppose the batch job is to use the functions INITIAL, COMPUTE, and DISPLAY from the workspace MYFNS. Assuming an appropriate MEMO file has been prepared, it could be checked by typing it out:

```
cp term apl on

TYPE INPUT1 MEMO

     )LOAD MYFNS
     INITIAL
     COMPUTE
     DISPLAY

CP TERM APL OFF
```

The job is sent to the batch machine by:

```
batchapl smith 999 test1 input1 myfns
```

The user should consult the VM/370: Command Language Guide for General Users for restrictions on the CP commands available under batch. The memo file must not contain an *)OFF* command; the BATCHAPL EXEC will supply the proper termination sequence. If the CMS editor is used to prepare the MEMO file then the IMAGE OFF option of the editor should be used. The BATCHAPL EXEC is adequate for many batch jobs, however it could be modified by your VM/370 system staff to allow additional workspaces and input files to be used.

SECTION 4: APL/CMS AUXILIARY PROCESSORS

The APL/CMS system includes the following auxiliary processors:

```
AP100   COMMAND
AP101   STACK INPUT
AP110   CMS DISK I/O
AP111   FILEDEF I/O
```

The COMMAND processor enables an APL/CMS user to issue CP and CMS commands. The STACK INPUT processor stores data for input at the first opportunity to APL/CMS. The CMS DISK I/O processor provides sequential and random access using the CMS file system. The FILEDEF I/O processor provides sequential access to devices supported by the OS simulation facilities of CMS available through the FILEDEF command using QSAM.

The example introduced in Section 2, "Using the APL/CMS Auxiliary Processors", is given in more detail below; refer to it as you read the following sections.

```
      S←'EXAMPLE SCRIPT(370'                SET INITIAL
      S                                     VALUE AND
EXAMPLE SCRIPT(370                          DISPLAY IT.

      110 ⎕SVO'S'                           SHARE VARIABLE
2                                           ... SHARE OK

      S                                     DISPLAY RESPONSE
0 1 1                                       TO INITIAL VALUE.

      S←'THIS GOES IN THE FIRST RECORD'     SET A VALUE
      S←'AND THIS GOES IN THE SECOND'       SET ANOTHER

⍝ THE FILE NOW CONTAINS TWO RECORDS.

      S                                     REFERENCE
THIS GOES IN THE FIRST RECORD               AND DISPLAY
      S                                     FIRST TWO
AND THIS GOES IN THE SECOND                 RECORDS.

      ⎕SVR'S'                               RETRACT VARIABLE
2                                           TO CLOSE FILE.
```

Remember that APL/CMS is an interactive system. The best method of learning to use the auxiliary processors is to get on the system and experiment. Try the previous example. The auxiliary processors are used within several workspaces in library 1: APFNS, CONVERT CPWSVMWS, and PRINT, for example.

INITIAL VALUE

When an offer to share with an APL/CMS auxiliary processor is made, the value of the variable being offered should be a character vector specifying the argument and options (if any) required by that auxiliary processor.

The general format for all initial values is:

```
'argument ( options'
```

The auxiliary processors supplied with APL/CMS assist the APL user in the transmission of data to a destination and the receipt of data from a source. The argument passed in the initial value is used to determine this source and/or destination.

Following the source or destination argument is a left parenthesis used to indicate "options follow". It should be present only when options are specified by the user. The APL/CMS auxiliary processors ignore extra left parentheses and disallow right parentheses. The interpretation of the initial value is covered later for each auxiliary processor.

OFFER PROTOCOL

The APL/CMS auxiliary processors match all shared variable offers by the user with counteroffers. The initial value of the shared variable is interpreted by the auxiliary processor as a user request. After inspecting the operating environment, the auxiliary processor specifies a new value for the shared variable. This value, when referenced by the user, will be a scalar 1 if the request is rejected. A new value of the variable can then be specified by the user after which the auxiliary processor (AP) repeats this procedure.

When an acceptable initial value is specified, the AP sets the shared variable to a scalar zero or a vector with the first element zero. After the user references this

value, the effect of all subsequent references and specifications of this shared variable is to move data and control information between the APL program and the auxiliary processor, as described for each processor. To specify a different argument or options for a shared variable, that variable must be retracted and reoffered with the required initial value.

OPTIONS FOR DATA CONVERSION

Data transmitted between the APL program and the auxiliary processors can be in three distinct forms:

1. APL variables, complete with size, shape, and type information. This is the most convenient and efficient form for most applications. The conversion option for this type of data is VAR.

2. Character vectors. This form is used primarily for interchange with other non-APL processors. Two conversion options, APL and 370, are described below.

3. Bit vectors (that is, zeros and ones) that provide the most general form of data transmission and interchange. The conversion option for this type of data is BIT.

Characters outside of the workspace (for example, data file records, punched cards, and printer lines) are transferrable, as characters, to and from the workspace in two ways:

1. Characters accepted by APL as input (processed as though entered from a keyboard) and produced by APL as output (as though for typing on a terminal). This option is called APL.

2. EBCDIC codes used by the System/370. This option is called 370.

You must have some knowledge of these two forms in order to transmit character data to and from external media. Some general information follows, with details in "Appendix A: Auxiliary Processor Conversion Options."

The APL script conversion option (APL) produces characters in the workspace as though the input data were entered from the keyboard; output data is created as though the characters in the workspace were typed at the terminal. For example, the character "A" in the workspace is converted to an "A", and the character "A" in the workspace is converted to the three characters: "A", "backspace", and "_".

The System/370 EBCDIC conversion option (370) provides a direct mapping between some APL characters and some EBCDIC characters. For example, the character "A" in the APL workspace converts to the EBCDIC "A", and the APL character "A" converts to EBCDIC "a".

INPUT/OUTPUT PROCESSING

Introduction

The APL/CMS system includes two auxiliary processors that provide file input and output capabilities. AP110, CMS DISK I/O, supports the CMS file system which allows both sequential and random accessing. AP110 creates files with fixed or variable length unblocked records. Random access of variable length records is inefficient compared to random access of fixed length records. AP110 processes blocked or unblocked fixed length records and unblocked variable length records.

AP111, FILEDEF I/O, may be used for sequential access to CMS disk files and other types of devices such as magnetic tape, card readers, line printers, and terminals. This device independence is achieved by using the OS simulation facilities in CMS. The AP111 user must issue an appropriate FILEDEF command to CMS (using AP100) before the specified data set can be opened. AP111 supports QSAM so that blocked or unblocked and fixed or variable length records can be sequentially processed. The workspace 1 APFNS contains functions which simplify the use of AP111. One function, for example, will automatically issue appropriate FILEDEF commands required in common situations. Load the workspace and type DESCRIBE.

CMS disk files with fixed length records can be processed by either processor regardless of blocking. Both I/O processors open files for reading and writing although switching between read and write is time-consuming.

Record Variables

An APL variable used to transmit data records is called a record variable. Except for the initial reference after properly specifying an initial value, all references of the record variable yield records from the file being processed. All specifications into the record variable cause records to be written into the file. The previous example used 'S' as a record variable.

Continuing the previous example:

```
      S←'EXAMPLE SCRIPT(370'   SET INITIAL VALUE
                               FOR RECORD VARIABLE.
      110 ⎕SVO'S'              SHARE VARIABLE
2                              ... SHARE OK

      S                        DISPLAY RESPONSE
0 1 3                          TO INITIAL VALUE.
```

The file just opened has two records; the 0 indicates acceptance of the initial value by AP110, the 1 is the position of the read pointer at the first record, the 3 is the position of the write pointer following the last record of the file.

Control Variables

An APL variable used to control or monitor data transmission is called a control variable. It is paired with the most recently offered, but unpaired, record variable specifying the same file or ddname. A control variable may be required to query certain status information. For AP110, it is required to achieve indexed selection of records from the file.

Except for the first reference after a proper initial value has been specified, the reference of a control variable returns a scalar (AP111) or vector (AP110) whose first element indicates the status of the previous specification or reference of its paired record variable. A zero indicates successful completion. For return codes and status indicators, see "Appendix B: Auxiliary Processor Return Codes."

The use of a control variable is illustrated by continuing the example:

```
          C←'EXAMPLE SCRIPT (CTL'  SET INITIAL VALUE
                                   FOR CONTROL VARIABLE.
          110 ⎕SVO'C'              SHARE VARIABLE
     2                             ... SHARE OK

          C                        DISPLAY RESPONSE
     0                             TO INITIAL VALUE.
```

The '0' indicates acceptance of the initial value by AP110.

Record Pointers

AP110 maintains both a read pointer and a write pointer. These pointers are independent and they indicate the position of the record to be processed by a reference or specification of the record variable. The value of the control variable is the status indicator followed by the read and write pointers. AP110 initializes these record pointers to 1 and N+1, where N is the number of records in the file, when the file is opened. The initial reference of the record variable returns a zero followed by these pointers.

Continuing our previous example:

```
          C                        DISPLAY CONTROL
     0 1 3                         VARIABLE.
```

AP110 increments the read or write pointer by one after each successful read or write of the file. Record pointers can be reset by the user at any time by specifying an integer into the control variable. A scalar sets the read pointer; a two-element vector sets both pointers. An integer of less than one does not change the pointer.

Continuing our previous example:

```
          A←S                      READ RECORD.
          C                        DISPLAY CONTROL
     0 2 3                         VARIABLE.
          S←Y                      WRITE RECORD.
          C                        DISPLAY CONTROL
     0 2 4                         VARIABLE.
          S←Z                      WRITE RECORD.
          B←S                      READ RECORD.
```

```
      C                          DISPLAY CONTROL
0 3 5                            VARIABLE.

      C←2                        RESET READ POINTER.
      C                          DISPLAY CONTROL
0 2 5                            VARIABLE.
      S                          READ RECORD.
AND THIS GOES IN THE SECOND

      ⎕SVR 2 1⍴'SC'              RETRACT VARIABLES.
2 2
```

Now A and B contain the first two records from EXAMPLE SCRIPT. Records 3 and 4 contain the variables Y and Z (assuming these variables were character vectors since the 370 conversion option was specified).

AP111 processes files sequentially and does not support record pointers. The FILEDEF command, as an option, sets both read and write pointers to the top of a file (the default) or to the bottom of a file (DISP MOD). This setting of the record pointers is in effect every time a file is opened for read, write, or switching between read and write. AP111 does not explicitly alter the position of a file (magnetic tape, for example) when the record variable is retracted.

End File and Error Conditions

Whenever you reference a record variable and an end of file is read, the I/O processor assigns a null vector to the variable. This is also done if a read error occurs. These cases can be differentiated by inspecting the return code available via a control variable. Null variables can be written only with the VAR option.

Space Used by Auxiliary Processors

INPUT/OUTPUT BUFFERS: AP110 and AP111 need virtual storage space for input/output buffers. This space is located outside the workspace in an area whose size is fixed immediately after APL is invoked. The standard size is 8192 bytes. If the auxiliary processors are used to transmit long records or to access many files simultaneously, they may fail to find buffer space and will post an error in the control variable.

The size of the buffer space can be set by supplying an argument when APL is invoked, thus:

```
ipl apl parm x

   -- or --

ipl cms
apl x
```

where:

x — is the number of bytes (for example, 4096) or the number of kilobytes (for example, 12K) or the number of megabytes (for example, 2M).

parm — is a necessary part of the IPL command.

The amount of space allocated is that requested, rounded up to the next page boundary. If the space is not available, a)OFF HOLD is issued.

PROGRAM STORAGE: Many CMS commands require some program storage space in which to operate. The size of this area varies greatly among commands. No such space is required for CP commands.

AP100 is used to invoke CP and CMS commands. If the APL/CMS system, as distributed, is invoked by an IPL command, then about 57,000 bytes of program storage are available for CMS commands called by AP100. In general, commands which require more program storage than is available will fail because the CMS storage management system should not allocate storage used by APL/CMS to the command; no harm should befall the APL/CMS system.

This is not the case if APL/CMS has been invoked as a command under CMS. The CMS storage management system will allocate space used by APL/CMS to any command which needs it. This means that any CMS command invoked via AP100 needing program storage space will cause an abrupt termination of APL/CMS. CMS commands available in the subset mode do not require program memory. The 1 APFNS workspace includes a function to temporarily invoke the CMS SUBSET mode which allows direct interactive CMS usage. An expert should be consulted if you wish to explore this area. Details are supplied in the APL/CMS Installation Manual.

AP100--THE COMMAND PROCESSOR

Initial Value

env
CMS
CP

Description

The operating environment available to the user of APL/CMS includes the environments of CMS and CP. Commands can be processed by CMS and CP to dynamically change the characteristics of these two environments. Character vectors, or one-element arrays, when specified into the shared variable, are immediately processed by the selected environment. Notice that the CMS environment includes a command, CP, that passes the rest of the arguments to CP. Subsequent references of the shared variable yield the return code set by CP or CMS. For details on return codes, see "Appendix B: Auxiliary Processor Return Codes". CP and CMS commands are described in the VM/370: Command Language Guide for General Users.

The commands destined for CMS are broken into "tokens". A token is a parenthesis or a series of nonblank characters. Only the first eight characters of each token are used. For both CP and CMS, all characters are converted via the 370 option immediately prior to transmission (see "Appendix A: Auxiliary Processor Conversion Options").

Warning: Some commands may cause abrupt failure of APL/CMS and loss of the active workspace; refer to "Space Used by Auxiliary Processors". The CP command to define the size of virtual machine storage or to IPL is an example of a disasterous command under all circumstances.

Argument

env
specifies the command environment and defaults to CMS.

Examples:

The function below requests multiple copies of any printed output.

```
      ∇ COPIES N;X
[1]     X←'CP'                      SET INITIAL VALUE
[2]     100 ⎕SVO'X'                 SHARE VARIABLE
[3]     X←'SPOOL PRT COPY ',⍕N      ISSUE COMMAND
      ∇
```

The function below will erase the file specified by the character right argument.

```
      ∇ Z←ERASE R;CMS
 [1]    Z←100 ⎕SVO CMS←'CMS'
 [2]    CMS←'ERASE ',R
 [3]    Z←CMS
      ∇
```

The value returned is 0 if the file is erased. A non-zero value would indicate an error condition; the value 28 indicates an nonexistent file.

AP101--THE STACK INPUT PROCESSOR

Initial Value

```
stk ( cvt ord
CMS   370 END
      APL BEG
          LIFO
          FIFO

   --or--

stk
APL
```

Description

Character vectors can be stored for subsequent input at the first opportunity to CMS and APL/CMS. Two areas are available. The first, in storage and used by CMS and APL, is called the CMS input stack; the second, a disk file used only by APL, is called the APL input stack. The CMS stack is efficient to use but limited by available storage; the APL stack is limited only by disk storage. Only character vectors (or one-element arrays) can be stored by this processor. A reference of the shared variable obtains the return code set by the last specification; zero indicates success. For other return codes, see "Appendix B: Auxiliary Processor Return Codes."

When CMS or APL/CMS issues a request for keyboard input, a value from the beginning of the stack is used and no entry is requested from the user. This entry is made as though the user backspaces to the left margin, strikes the attention, and enters the value. The CMS stack has priority over the APL stack. If the user generates an APL interrupt or if a character error is detected by APL/CMS when using stacked input, both input stack areas are flushed and the keyboard unlocks for input.

Warning: Certain values such as HX, HT, and RT cause immediate action by CMS (and are not actually stacked) when placed in the CMS stack. Refer to the section on "Immediate Commands" in the VM/370: Command Language Guide for General Users.

Argument and Options

stk
specifies the input stack to be used. The default is CMS.

cvt
is the standard option for character translation and defaults to 370. This option may be used only with the CMS stack. The APL stack is maintained in an internal code which requires no conversion.

ord
indicates whether the processor places data at the beginning of the stack (BEG or LIFO) or at the end (END or FIFO). The default is END. This option may be used only with the CMS stack. Entries into the APL stack always use the FIFO method.

Example:

This function will save the active workspace and return. The method used is to place a)SAVE CONTINUE command and a branch to the statement RESUME at the beginning of the CMS stack. The CMS stack has priority over the APL stack and the BEG option will place the command in front of anything already in that stack. Execution is suspended by setting the stop vector for a stop on statement with the label RESUME. The stack entry is read at this point which saves the workspace. The stack is read again whereupon the branch causes execution to be resumed. The special CMS immediate commands, HT and RT, are used to prevent the normal terminal output.

```
     ∇ CHECKPOINT;S;T
[1]    S←'CMS (APL BEG'          SELECT CMS STACK, LIFO
[2]    T←101 ⎕SVO'S'             SHARE S, IGNORE RESULT
[3]    S←'HT'                    HALT TERMINAL OUTPUT
[4]    S←'→RESUME'               SECOND OUT OF STACK
[5]    S←')SAVE CONTINUE'        FIRST OUT OF STACK
[6]    S⍙CHECKPOINT←RESUME       SET STOP VECTOR
[7]  RESUME:S←'RT'               RESUME TERMINAL OUTPUT
     ∇
```

AP110--THE CMS DISK I/O PROCESSOR

Initial Value

```
nam ( cvt fmt
      VAR FIX
      APL
      370
      BIT

   --or--

nam ( typ
      CTL
```

Description

This processor provides sequential and random access to disk files under control of the CMS file system. The operation of this processor is described in the section on input/output file processing. The CMS disk file system is described in the VM/370: Command Language Guide for General Users.

Argument and Options

nam
specifies the name of the CMS disk file to be accessed. It has one of the forms:

```
filename
filename    filetype
filename    filetype    filemode
```

The default filemode is A1. The default filetype is VMAPLcF, where c is the first character of the conversion method used.

cvt
specifies the standard option for conversion which defaults to VAR except when the user gives a filetype of VMAPLcF. In this case, the default will be the cvt option with matching first letter. For example, if the user gives VMAPL3F as the filetype, then the default conversion is 370.

Note: All combinations of filetype and conversion can be explicitly specified. The 370 conversion option is used for filename, filetype, and filemode.

fmt
indicates the type of file to be created. The default is a file with variable length records. When FIX is specified, the file will contain fixed length records; the record length is the length of the first record written into the file. Each subsequent record must have this same length or an error is reported (in the control variable).

Note: If the file already exists when the offer is made, the FIX option is ignored and the existing file characteristics are used.

typ
establishes a control variable for the file, if an unpaired record variable already exists for the same file. The cvt and fmt options may be given but are ignored if the CTL option is present.

AP111--THE FILEDEF I/O PROCESSOR

Initial Value

```
ddn ( cvt
      VAR
      APL
      370
      BIT

   --or--

ddn ( typ
      CTL
```

Description

This processor provides sequential access, via QSAM, to any device supported by VM/370. The device and its characteristics are specified by use of the CMS command, FILEDEF. The operation of this processor is described in the section on input/output file processing. The FILEDEF command is described in the VM/370: Command Language Guide for General Users. The workspace 1 APFNS contains functions to facilitate AP111 usage.

Argument and Options

ddn
  specifies the ddname to be accessed. It must be the ddname defined by a FILEDEF command that the APL user has already issued to CMS.

cvt
  specifies the standard option for conversion and defaults to VAR.

typ
  establishes a control variable for the file, if an unpaired record variable already exists for the same ddname. The cvt option may be given but is ignored if the CTL option is present.

EXAMPLES USING THE INPUT/OUTPUT PROCESSORS

Examples for AP110

Example 1 uses CMS disk files for a sequential update function. SUP is given the name of a CMS file as an argument. It updates "name VMAPLAF A1" using "name CHANGES A1" and creating "name WORKFIL A1" as a temporary new file. When the update is complete, SUP erases the old file and renames the new file. The files SUP processes contain personnel records that are identified by the person's social security number as the first nine characters. The changes file consists of complete replacement records or, if the record is to be deleted, merely the social security number. SUP provides a return code of 0=successful completion, 1=update performed but no file erased or renamed, or 2=nothing done.

Example 1:

```
      ∇ Z←SUP FILENAME;OLDFIL;OLDREC;OLDSEQ;CNGFIL;
            CNGREC;CNGSEQ;NEWFIL;NUMS;CMS;⎕IO
[1]     Z←2
[2]     OLDFIL←FILENAME,' VMAPLAF'
[3]     CNGFIL←FILENAME,' CHANGES (APL'
[4]     NEWFIL←FILENAME,' WORKFIL (APL'
[5]     110 ⎕SVO'OLDFIL'
[6]     110 ⎕SVO'CNGFIL'
[7]     110 ⎕SVO'NEWFIL'
[8]     →(9≠⍴OLDFIL,CNGFIL,NEWFIL)/⎕IO←0
[9]    'CNGREC←9↑NUMS←'0123456789'
[10]    →STRT
[11]  USEO:NEWFIL←OLDREC
[12]    OLDSEQ←10⊥NUMS⍳9↑OLDREC←OLDFIL
[13]  LOOP:→(OLDSEQ<CNGSEQ)/USEO
[14]    →(OLDSEQ>CNGSEQ)/USEC
[15]    →(0=⍴OLDREC,CNGREC)/ENDF
[16]  STRT:OLDSEQ←10⊥NUMS⍳9↑OLDREC←OLDFIL
[17]  USEC:→(9=⍴CNGREC)/DLET
[18]    NEWFIL←CNGREC
[19]  DLET:CNGSEQ←10⊥NUMS⍳9↑CNGREC←CNGFIL
[20]    →LOOP
[21]  ENDF:⎕SVR 3 6⍴'OLDFILCNGFILNEWFIL'
[22]    100 ⎕SVO'CMS'
[23]    →(Z←CMS)/0
[24]    CMS←'ERASE ',FILENAME,' VMAPLAF'
[25]    CMS←'RENAME ',FILENAME,' WORKFIL A1 = VMAPLAF ='
      ∇
```

Example 2 illustrates the CMS disk file random access. The FIND function is given a personnel file, such as that updated by SUP, and a social security number. It is to find the location of the corresponding record (if any) in the file. In particular, if it returns an integer (n), then the nth record has the matching social security number. If it returns a real (n.5), then the given social security number is not in the file. If it were, it would occur between records n and n+1 (0 <= n <= number of records in the file). The only other possible return value is "SHARE ERROR" which indicates that the FIND command could not establish the necessary shares with AP110.

Example 2:

```
     ∇ Z←FILE FIND SS;REC;NUM;⎕IO;BOT;TOP;ID
[1]    REC←FILE,' VMAPLAF'
[2]    NUM←REC,'(CTL'
[3]    110 ⎕SVO'REC'
[4]    110 ⎕SVO'NUM'
[5]    →(NUM∨1↑REC)/FAIL
[6]    Z←2*⌈2⍟¯1↑NUM
[7]    ⎕IO←BOT←0
[8]  DROP:TOP←Z
[9]  LOOP:Z←NUM←0.5×BOT+TOP
[10]   →(Z≠⌈Z)/0
[11]   ID←10⊥'0123456789'⍳9↑REC
[12]   →(SS<ID)/DROP
[13]   BOT←Z
[14]   →LOOP×SS>ID
[15] FAIL:Z←'SHARE ERROR'
     ∇
```

Example 3 assumes that a function is suspended with a domain error. The function was using AP110 to sequentially read and process INPUT DATA and the domain error occurred because the latest record read had alphabetic characters where a numeric field was supposed to be. To locate where in the file the bad record occurs, use the following sequence, using 0-origin, to determine its record number:

Example 3:

```
      X←'INPUT DATA ( CTL'
      110 ⎕SVO'X'
2
      X
0
      BAD←¯1+X[1]
```

Examples for AP111

Example 4 uses the unit record equipment for a card-to-printer function. The program, CTOP, expects as input a series of decks stacked in the card reader as a single file. These cards contain a sequence field in the last four columns and a deck identification code in the preceding four columns. The cards are to be listed on the printer and each deck should start on a fresh page.

Example 4:

```
     ∇ CTOP;CMS;SVPRINT;SVREAD;ID;CARD
[1]    100 ⎕SVO'CMS'
[2]    CMS←'FILEDEF CTOPOUT PRINTER (RECFM VA BLKSIZE 137'
[3]    CMS←'FILEDEF CTOPIN READER (RECFM F BLKSIZE 80'
[4]    SVPRINT←'CTOPOUT ( 370'
[5]    SVREAD←'CTOPIN ( 370'
[6]    111 ⎕SVO 2 7 ⍴'SVPRINTSVREAD '
[7]    →(SVPRINT∨SVREAD)/0
[8]    ID←'~'
[9]  LOOP:→(0=⍴CARD←SVREAD)/0
[10]   →(∨/ID≠4↑¯8↑CARD)/SKIP
[11]   SVPRINT←' ',CARD
[12]   →LOOP
[13] SKIP:ID←4↑¯8↑CARD
[14]   SVPRINT←'1',CARD
[15]   →LOOP
     ∇
```

The function shown in Example 4 would take care of many card-to-printer tasks. However, because it uses characters, the input is converted from 370 to APL and the output is converted back from APL to 370. Thus, for example, CTOP will not print cent symbols (¢).

Example 5 is a card-to-card function that has the same conversion problem as example 4. In fact, its conversion problem is even more acute since CTOC might be used to reproduce text decks. CTOC solves the problem by using BIT conversion. As input, CTOC expects a series of decks stacked in the card reader as separate files. The first card of each deck has an identification code in columns one to four. The remainder of the deck is to be reproduced except for the last eight columns of each card. The last eight output columns are to contain the deck identification code and a sequence number.

Example 5:

```
     ∇ CTOC FILES;⎕IO;BITS;PUN;RDR;NUM;ID;CARD;SEQ;F
[1]    ⎕IO←0
[2]    BITS←(10 4⍴1),⍉2 2 2 2⊤⍳10
[3]    100 ⎕SVO'F'
[4]    F←'FILEDEF FILEO PUNCH(RECFM F BLKSIZE 80'
[5]    F←'FILEDEF FILEI READER(RECFM F BLKSIZE 80'
[6]   BEGF:PUN←'FILEO(BIT'
[7]    RDR←'FILEI(BIT'
[8]    111 ⎕SVO 2 3⍴'PUNRDR'
[9]    →(PUN∨RDR)/NUM←0
[10]   ID←32↑RDR
[11]  LOOP:→(0=⍴CARD←RDR)/ENDF
[12]   SEQ←,BITS[10 10 10 10⊤NUM←NUM+10;]
[13]   PUN←(¯64↓CARD),ID,SEQ
[14]   →LOOP
[15]  ENDF:⎕SVR'PUN'
[16]   ⎕SVR'RDR'
[17]   →(0<FILES←FILES-1)/BEGF
     ∇
```

MULTIPLE ACCESSING

The preceding examples have been in terms of single variables:

1. How do you read a file?

2. How do you send commands to CP via a shared variable?

Let us consider the implications of the ing of multiple variables with an auxiliary processor.

The auxiliary processors, COMMAND and STACK INPUT, can share multiple variables with different or identical destinations. Consider the following example:

Example 6:

```
      A←B←C←'CP'
      100 ⎕SVO 3 1⍴'ABC'
2 2 2
      A←'INCORRECT REQUEST'
      B←'SPOOL 00E OFF'
      C←'Q F'
FILES:  NO RDR,  NO PRT,  NO PUN
```

The variables A, B, and C in Example 6 are independent, yet they have something in common. No one of them can affect any of the others. If you reference A, you obtain the return code that indicates an invalid CP command. Using B and C to execute successful commands has not changed this return code. In common, they share the same destination, so their assignments are "merged"; they are all sent to CP. For COMMMAND and STACK INPUT this is not significant, but for other auxiliary processors it is.

The I/O auxiliary processors accept multiple record variables; thus, they allow simultaneous access to several files. Example 1 for AP110 is an update function that reads both the old file and the changes file. The I/O auxiliary processors also accept multiple record variables with identical data sources or destinations. Access to the same file with multiple variables can be very useful, (to use more than one conversion method, for example) although it may be confusing. The following examples explore this situation.

If A and B are both shared variables using the card reader as their source and you reference A, B, and finally A again, then the second reference of A does not read the second card. Rather, its value is for the third card since the second card was read by the reference to B. Independent variables causing a merged effect can allow, for example, any APL function to print information without knowing whether or not some higher calling function is also printing.

Now assume that A and B are both shared with the APL/CMS DISK I/O processor and that both had the initial value "SOMEFILE". If one references A, then B, and finally A again, then the second reference of A reads the second record. The first B reference and the first A reference each obtain the first record. These A and B accesses are independent.

Assume that A and B are both shared with the FILEDEF I/O processor and that both are using the same CMS disk file. If they are using different ddnames (for example, different

FILEDEF commands were issued each specifying the same CMS disk file) then A and B read records independently. The APL user should not attempt multiple access while the file is being created by AP111.

The I/O processors also allow multiple control variables. Because a control variable is accepted only if there is an unpaired record variable for the corresponding file, the meaning of these multiple control variables is clear. However, one should be aware of the method used for matching control variables with record variables. The control variable will be paired with the most recently shared unpaired record variable for the corresponding file (see Example 7).

Example 7:

```
     ∇ PROG;DATA;NUMB
[1]    NUMB←(DATA←'FT70 FILE'),'(CTL'
[2]    110 ⎕SVO'DATA'
[3]    110 ⎕SVO'NUMB'
          .
          .
          .
```

Example 7 shows that NUMB is paired with DATA. There is no problem if some higher calling function is also reading FT70 FILE without a control variable. Now consider Example 8.

Example 8:

```
     ∇ F1 COMPAR F2;R1;I1;R2;I2
[1]    I1←(R1←F1),'(CTL'
[2]    I2←(R2←F2),'(CTL'
[3]    110 ⎕SVO'R1'
[4]    110 ⎕SVO'R2'
[5]    110 ⎕SVO'I1'
[6]    110 ⎕SVO'I2'
          .
          .
          .
```

In most cases, the COMPAR function will work. However, if the fileid in F2 is the same as that in F1, then I1 is paired with R2 and I2 is paired with R1. To avoid malfunction, statements 4 and 5 should be reversed.

OTHER AUXILIARY PROCESSOR DETAILS

All initial values are converted using the 370 option before they are inspected, thus allowing you to refer to filenames that include 370 characters such as the $ (see "Appendix A: Auxiliary Processor Conversion Options" for details).

Options can occur in any order. If conflicting options occur (for example, 370 and BIT), then the option selected depends on the auxiliary processor. Blanks can be used freely: the initial value can use or omit leading or trailing blanks. The 'options follow' left parenthesis can occur with or without a preceeding or following blank. Any blank can be replaced by multiple blanks.

In some cases, records must be changed in length. When made longer, the process is known as padding; the elements added as a result are called pad characters. The APL/CMS auxiliary processors pad records on occasion. When this is necessary, character records are padded with blanks and bit records are padded with zeros. Records using VAR conversion are never padded. BIT records may require padding, even if they are of variable format, due to hardware limitations. On the IBM System/370, for example, all BIT records must have a length that is some multiple of eight. There is no case in which character records must be padded.

Note: The APL/CMS system includes a workspace called 1 APFNS. This workspace contains functions that facilitate usage of the auxiliary processors. For example, one function issues appropriate FILEDEF commands and offers shared variables to AP111. For details, load this workspace and type DESCRIBE.

The CMS auxiliary processors provide conversion to and from the workspace. The details of the conversion are given below.

THE 370 CONVERSION OPTION

Many characters are common to both the APL and EBCDIC character sets. The conversion preserves most of these characters. These characters are the same in both sets:

A THROUGH Z   0 *THROUGH* 9   *SPACE*
<   =   >   +   −   *
'   .   :   ;   ,   ?   !
(   )   \ | /   _

These characters have different graphics:

APL:	*A THROUGH Z* (underscored)	~	∧	¨	α	ǂ	≠	Δ	⊖	⌽
370:	a THROUGH z	¬	&	"	@	%	$	#	−0	+0

(Note that +0 and −0 are the EBCDIC left and right braces.) For example, "A" (underscored) is converted to "a" on output and "a" is converted to "A" (underscored) on input.

The following conversion occurs only when going from APL to 370.

APL:	←	¯
370:	=	−

The terminal control characters backspace, horizontal tabulate, idle, line feed, and new line, are translated one for one. All other EBCDIC codes are converted to "o" when translated to APL. All other APL characters are converted to a space when going to 370. Those with graphics are:

⌶   ⊥   ⊤   ⍝   ⍕
[   ]   ⌈   ⌊   ⍀   ⌿
○   ⍉   ⍟   ∘
⊂   ⊃   ∩   ∪   A

∊   ⍳   ⍴   ⍵
∨   ⍲   ⍱   ≤   ≥   ×
→   ↑   ↓   ⍋   ⍒
⌹   ⍞   ⎕   ∇   ⍫   ⍙

THE APL CONVERSION OPTION

Figure 1 shows the EBCDIC code as decimal integers, with corresponding APL graphics. This table is indicative of the conversion done by VM/370 when the APL type element is specified. The full APL character set is formed by use of the backspace (BS) terminal control code in conjunction with the other characters. For example, "A" is converted to "A", backspace, "_" on output and "_", backspace, "A" is converted to A on input.

Output to and input from files are both converted by the APL/CMS system as if going to or coming from the normal APL terminal. For input, all characters not in the APL/CMS input character set below, such as invalid codes and invalid compound characters (that is, those producing a character error on keyboard entry) are converted to one unique internal APL code, with no graphic, which CP normally prints as a space. For output, all characters other than the APL/CMS output character set below, are converted to the EBCDIC code 0.

The Input Character Set of APL/CMS:

The input character set comprises the following:

ABCDEFGHIJKLMNOPQRSTUVWXYZΔ
ABCDEFGHIJKLMNOPQRSTUVWXYZΔ
0123456789( )[ ]⌈⌊⍀\|/⌿_⍙⊥⌶⍎
⍺⍷⍳⍴⍵¯−+÷×*⍱∧⍲⍵<≤=≥>≠←→↑↓⍋⍒
⍟⌽⍉⊖⍟⍬∩∪⊂⊃∇⍫~⍁⍂'⍎.:"⍝;,?

and SPACE.

The Output Character Set of APL/CMS:

The output character set comprises the input character set, above, plus the terminal control characters:

BS - backspace
HT - horizontal tabulate
IL - idle
LF - line feed
NL - new line

370	APL	370	APL	370	APL	370	APL	370	APL	370	APL	370	APL	370	APL
0		32		64	*SPACE*	96	-	128		160		192		224	
1		33		65	"	97	/	129		161		193	*A*	225	
2		34		66	‾	98	→	130		162		194	*B*	226	*S*
3		35		67	≤	99	⍺	131		163		195	*C*	227	*T*
4		36		68	≥	100	⌈	132		164		196	*D*	228	*U*
5		37		69	≠	101	⌊	133		165		197	*E*	229	*V*
6		38		70	∨	102	∇	134		166		198	*F*	230	*W*
7		39		71	∧	103	∆	135		167		199	*G*	231	*X*
8		40		72	⍎	104	∘	136		168		200	*H*	232	*Y*
9		41		73	×	105	⍞	137		169		201	*I*	233	*Z*
10		42		74		106	[	138		170		202		234	
11		43		75	.	107	,	139		171		203		235	
12		44		76	<	108		140		172		204		236	
13		45		77	(	109	_	141		173		205		237	
14		46		78	+	110	>	142		174		206		238	
15		47		79	|	111	?	143		175		207		239	
16		48		80		112	]	144		176		208		240	0
17		49		81	⍵	113	⊂	145		177		209	*J*	241	1
18		50		82	∈	114	⊃	146		178		210	*K*	242	2
19		51		83	⍴	115	∩	147		179		211	*L*	243	3
20		52		84	~	116	∪	148		180		212	*M*	244	4
21		53		85	↑	117	⊥	149		181		213	*N*	245	5
22	*(BS)*	54		86	↓	118	⊤	150		182		214	*O*	246	6
23		55		87	⍳	119	\	151		183		215	*P*	247	7
24		56		88	○	120		152		184		216	*Q*	248	8
25		57		89	←	121		153		185		217	*R*	249	9
26		58		90		122	:	154		186		218		250	
27		59		91		123		155		187		219		251	
28		60		92	*	124		156		188		220		252	
29		61		93	)	125	'	157		189		221		253	
30		62		94	;	126	=	158		190		222		254	
31		63		95		127		159		191		223		255	

Figure 1. EBCDIC Codes (Integers) to APL/CMS Characters (Graphics) via APL Conversion Option

APPENDIX B: AUXILIARY PROCESSOR RETURN CODES

Section 4 indicates that the APL/CMS auxiliary processors provide return codes describing the results of a previous operation. With the COMMAND and STACK INPUT processors, you reference the shared variable to obtain the return code. The I/O processors provide return codes via a control-type shared variable.

The return codes from the COMMAND processor are generally the return code from the command previously assigned to the shared variable. Since you can issue any number of commands, including commands written by yourself, it is impossible to list all possible COMMAND return codes. A few errors are intercepted and result in return codes that have been generated by the auxiliary processor itself. If an I/O problem causes the CMS OS simulation routines to take a SYNAD exit, AP111 generates a decimal return code which, when converted to hexadecimal, has the four-byte representation N0,N1,A0,A1. The first two values are the two sense bytes and the latter two values are the two status bytes. All other return codes are generated by the auxiliary processors or result from some known CMS macro instruction. All specific return codes are given below.

Numerical Listing

-3 Unknown CMS command.

0 No error exists.

1 Attempt to read a nonexistent file or unknown CP command.

3 Permanent read error.

5 Attempt to write in a file with a bad filemode (second character) or to read a file with too many records for CMS.†

6 Attempt to write too many records in a CMS file.†

7 Attempt to read a record with invalid format or attempt to write past the end of a variable-length file. You can always write at the end of file (that is, you may append a record to the file). With a fixed-length file you may also write past this point and one or more blank records are inserted into the file; this is not possible with variable length files.

8 Attempt to read a record with incorrect record length from a file with fixed format.

10 Attempt to create a file when you already have the maximum allowed by CMS.†

12 End-of-file read or attempt to write on a read-only disk.

13 Attempt to write on a full disk.

---

†The VM/370: Command Language Guide for General Users gives these limits.

14 Attempt to write on an unformatted disk.

15 Attempt to write a record with incorrect length into a file with fixed format.

17 Attempt to write a record that is too large into a variable length file.†

19 Attempt to write in a file already containing as many data blocks as CMS will allow.†

440 Data set cannot be opened for output.

441 Data set cannot be opened for input.

442* ABEND from the CMS OS simulation routines.

443* Insufficient free storage for the CMS OS simulation routines.

444* You assigned an invalid value to a shared variable. This cannot happen with VAR conversion. The value is invalid because it is null or is an array, has the wrong type, or is too big. An error occurs, for example, if BIT conversion is being used and the value is 1 2 3. An error occurs with the APL and 370 conversion options if the value to be converted is numeric instead of character.

445 You referenced a shared variable that is reading a file using VAR conversion and the resulting record is not a valid APL variable in internal form. For example, it may not have a descriptor, the element count may not equal the times reduction of the shape vector, etc.

---

†The VM/370: Command Language Guide for General Users gives these limits.

*If these errors occur, you can restart APL/CMS with more free storage and try again. For details, see Section 4, heading "Space Used by Auxiliary Processors."

Return Codes by Processor

This information is included for the experienced VM/370 user of APL/CMS. Each auxiliary processor is listed with an indication of the origin of its return codes. Each processor can, in addition, return the 44x codes.

AP100

CP: Result Register after Diagnose.
CMS: Register 15 after SVC 202.

AP101

APL: Register 15 after FSWRITE
CMS: Register 15 after SVC 202 for the ATTN function.

AP110

Register 15 after FSREAD or FSWRITE

AP111

Sense/Status Information on SYNAD exit.
Errors 12, 15, 17

READER'S COMMENTS

Title: APL/CMS User's Manual
Programming RPQ MF2608
Program Number 5799-ALK

Order No. SC20-1846-1

Please check or fill in the items; adding explanations/comments in the space provided.

Which of the following terms best describes your job?

☐ Customer Engineer	☐ Manager	☐ Programmer	☐ Systems Analyst
☐ Engineer	☐ Mathematician	☐ Sales Representative	☐ Systems Engineer
☐ Instructor	☐ Operator	☐ Student/Trainee	☐ Other (explain below)

How did you use this publication?

☐ Introductory text ☐ Reference manual ☐ Student/ ☐ Instructor text
☐ Other (explain) _____

Did you find the material easy to read and understand? ☐ Yes ☐ No (explain below)

Did you find the material organized for convenient use? ☐ Yes ☐ No (explain below)

Specific criticisms (explain below)

Clarifications on pages _____
Additions on pages _____
Deletions on pages _____
Errors on pages _____

Explanations and other comments:

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

Trim Along This Line

YOUR COMMENTS PLEASE . . .

This manual is one of a series which serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your comments on the back of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

FOLD FOLD

FIRST CLASS
PERMIT NO. 38
PALO ALTO, CA.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

IBM SCIENTIFIC CENTER
APL/CMS Publications
2670 Hanover Street
Palo Alto, California 94304

FOLD FOLD

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

Trim Along This Line

APL/CMS User's Manual Programming RPQ MF2608 Printed in U.S.A. SC20-1846-1