SERIES 60 (LEVEL 64)

INTERACTIVE OPERATION FACILITY

SUBJECT

Description of the Interactive Operation Facility (IOF), Including Time Sharing Capabilities, Command Language, Text Editor Functions, Library Maintenance Aspects, and System Considerations

SOFTWARE SUPPORTED

GCOS Software Release 0400

ORDER NUMBER

AQ60, Rev. 0

September 1978

Honeywell

PREFACE

This manual describes the Interactive Operation Facility available with the GCOS operating system for Series 60 Level 64 computers. Familiarity with "GCOS Series 60 Level 64 Job Control Language Reference Manual" is assumed.

Section I introduces the general concepts and facilities available with IOF. Section II provides background information for IOF users. Section III describes the Interactive Command Language, Break Commands, and JCL Statements. Section IV cover the Text Editor and how to write Editor programs. Section V with Interactive Library Maintenance. Section VI describes IOF from a System viewpoint and is primarily intended for System Managers.

Each section of this document is structured according to the heading hierarchy shown below. Each heading indicates the relative level of the text which follows it.

Level	Heading Format
1 (highest)	ALL CAPITAL LETTERS, UNDERLINED
2	Initial Capital Letters, Underlined
3	ALL CAPITAL LETTERS, NOT UNDERLINED
4	Initial Capital Letters, Not Underlined
5 (lowest)	ALL CAPITAL LETTERS FOLLOWED BY COLON: Text begins on same line.

File No.: 1N33

LEVEL 64 DOCUMENT LIST

Order Number	Title
AQ02	*Series 100 Program Mode Operator Guide*
AQ03	*Series 100 Conversion Guide*
AQ04	*Series 200/2000 Conversion Guide*
AQ05	*System 360/370 Conversion Guide*
AQ09	*System Management Guide*
AQ10	*Job Control Language (JCL) Reference Manual*
AQ11	*Job Control Language (JCL) User Guide*
AQ13	*System Operation Operator Guide*
AQ14	*System Operation Console Messages*
AQ18	*Operator Reference Manual*
AQ20	*Data Management Utilities Manual*
AQ21	*Series 200/2000 Program Mode User Guide*
AQ22	*Series 200/2000 Program Mode Operator Guide*
AQ26	*Series 100 File Translator*
AQ27	*Series 200/2000 File Translator*
AQ28	*Library Management Manual*
AQ40	*System 3 Conversion Guide*
AQ49	*Network Control Terminal Operation Manual*
AQ50	*Terminal Operations Manual*
AQ52	*Program Checkout Facility Manual*
AQ53	*Communications Processing Facility Manual*
AQ55	*TDS/64 Standard Processor Site Manual*
AQ56	*TDS/64 User Guide*
AQ57	*Standard Processor Programmer Reference Manual*
AQ59	*Unit Record Devices User Guide*
AQ63	*COBOL User Guide*
AQ60	*Interactive Operation Facility*
AQ64	*COBOL Language Reference Manual*
AQ65	*FORTRAN Language Reference Manual*
AQ66	*FORTRAN User Guide*
AQ67	*FORTRAN Mathematical Library*
AQ68	*RPG Language Reference Manual*
AQ69	*RPG User Guide*
AQ72	*Series 200/2000 COBOL to Level 64 COBOL Translator*
AQ73	*IBM COBOL Translator*
AQ82	*BFAS User Guide*
AQ83	*HFAS User Guide*
AQ84	*UFAS User Guide*
AQ85	*Sort/Merge Manual*
AQ86	*Catalog Management Manual*
AQ87	*Library Maintenance User Guide*
AQ88	*I-D-S/II User Guide, Volume 1*
AQ89	*I-D-S/II User Guide, Volume 2*
AQ90	*COBOL Reference Card*
AQ92	*Operator's Reference Card*
AQ93	*RPG Reference Card*
AQ94	*FORTRAN Reference Card*

CONTENTS

SECTION I

INTRODUCTION

The Interactive Operation Facility (IOF) provides the GCOS user with a time-sharing capability. The scope of such a system is very large and may include different functions, such as interactive program preparation and execution, interactive file query and an interactive checkout facility to debug programs.

OVERVIEW OF IOF

The IOF system has been designed with a few basic objectives considered essential in the context of a modern operating system. The main features can be summarized as follows:

- Use of existing command languages. No effort is required to learn the new language and its capabilities. The user has to know the standard Job Control Language (JCL), the standard Operator Control Language (OCL) and the standard Library Maintenance command language. Syntax and semantics are identical whatever the mode of execution used: batch or interactive.

- Integrated Resource management and accounting. Both interactive and batch executions are handled the same way by the GCOS-64 system. Thus an interactive user session competes with a batch job, at the same level, for the same kind of system resources. Accounting information related to the use of resources is also recorded within the standard accounting file. Also the master operator can control the maximum number of logged-on users and can modify priorities of different interactive user sessions in addition to batch jobs.

- High availability of the IOF system. In particular, the abnormal termination of an interactive user session due to a user or a system program error has no effect on the other sessions in progress. In fact the visibility is the same as that which exists between batch jobs running concurrently.

- Good security. This ensures privacy protection at different levels from the log-on to the independence of memory address spaces.

FACILITIES AVAILABLE WITH IOF

IOF provides the following facilities:

- Program preparation: IOF can be used to develop and maintain source

programs (COBOL, FORTRAN, RPG) stored within disk libraries or sequential files.

- JCL preparation: IOF can be used to develop and maintain JCL procedures stored in disk libraries.
- Batch job submission as well as the ability to control job execution and output deliveries in the same way as the master operator.
- Scanning of output deliveries generated by the submitted jobs to check them before their final printout.

INTERACTIVE SESSIONS

An interactive session (or an interactive user session) is defined by the ordered set of commands that are entered by the user from his log-on to his log-off.

Basically an interactive session is equivalent to the execution of a job, the log-on being the $JOB statement and the log-off the $ENDJOB statement. So an interactive session has the same properties as a normal batch job, the only difference being that the JCL commands are interactively entered and executed. We will call the job associated with the interactive session an 'Interactive Session Job' or simply an 'Interactive Job'. The interactive job consists of successive executions of the IOF step which is the conversational interpreter of the system commands and of the different steps called by the interactive user. Hence, after log-on and the initiation of the associated interactive job, the first executed step is the IOF step, which monitors the system command level (i.e. JCL). Each time another system processor (such as the Library Maintenance Processor) is called by using a $STEP..$ENDSTEP description or an extended statement, the IOF step is terminated and the called step is loaded in its place, as in the case of a batch job. The IOF step will be reloaded at the completion of the called step in order to return the terminal to the system command level (JCL).

There is no stream reading or asynchronous input function, so the interactive job cannot have input enclosures. Commands and data are interactively entered, providing the respective program support and conversational mode. Commands and data may also be stored in library members or sequential files.

SYSTEM COMMAND

A system command is a request to the system that some function be performed. It may be an OCL (Operator Command Language) command to request some control or status about the system operation and particularly about the batch jobs submitted by the interactive user. It can also be a JCL (Job Control Language) statement at the job enclosure level, which may be either a control statement to the JCL translator (ex.LIB), or a program call (ex.LIBMAINT). The called program may be conversational and so communicates with the user's console. It could be, for example, the Library Maintenance Processor whose command language can be entered in conversational mode. See figure 1-1 which shows how IOF activates standard GCOS functions to execute JCL statements and call programs.

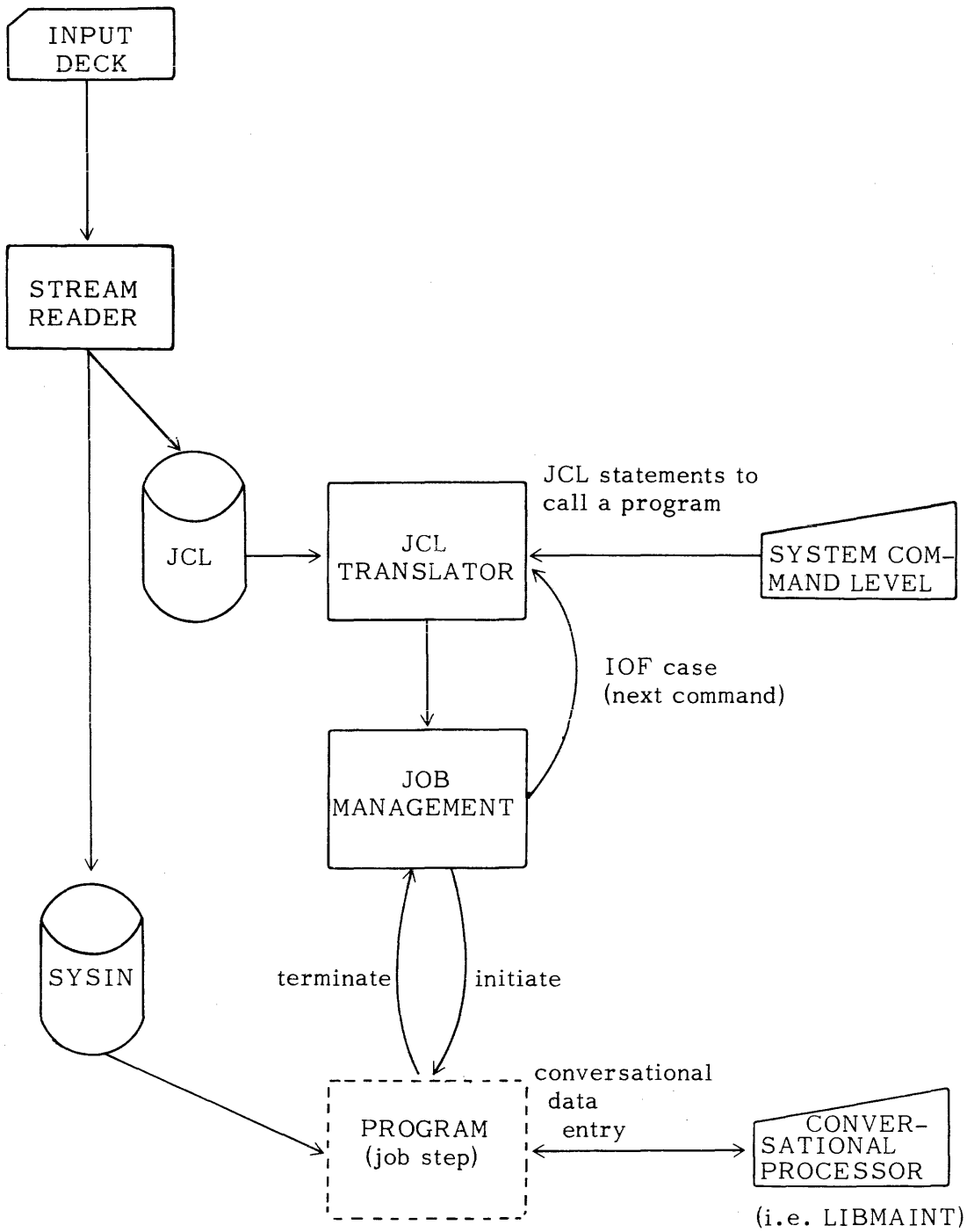


Figure 1-1 Interactive Program and Batch Program Execution.

SECTION II

INTERACTIVE OPERATION FROM USER VIEWPOINT.

This chapter describes the operation of a terminal from the interactive user's point of view. We shall first see the conditions that are necessary to allow a log-on under IOF. We shall then define the basic interface between the terminal and the system, promts, message editing, system messages, the communication mail boxes and text input. Once these concepts are well defined, we shall be able to describe the operation of an IOF terminal and consider a conversational program.

PROMPTS

A prompt means that the terminal is waiting for an input text. The prompt is aligned in the first column of the terminal line with the input text following up to maximum of 255 characters. Since there are different levels of conversational language it is useful to introduce the different types of prompts. The prompt provides information about the processing level of the terminal and in particular indicates which language the user is authorized to enter at that moment.

PROMPTS	PROCESSING LEVELS
S:	System Command level, only OCL and JCL languages allowed.
-:	Line Continuation, current line is not complete and terminal is waiting to continue.
C:	Processor Command level, in the case of the Library Maintenance processor (LIBMAINT), only the LIBMAINT language is authorized.
R:	Request level, in the case of the Library Maintenance processor only the Text Editor language is authorized.
I:	Input Data Request, the conversational program requests data. In the case of the Library Maintenance processor it may mean that you are within the Text Editor in an 'Append', 'Insert' or 'Change' sequence;
???	Break interruption.

MESSAGE EDITING

The IOF software adapts the message for the terminal's physical characteristics, the message is automatically split onto several lines, VIP-like terminals are handled in a no scrolling mode. This means the operator has to enter a ready signal (e.g. $*$ RDY BTNS Command) in order to unlock the screen when filled with messages and thus be able to display the following messages.

Message editing observes a number of conventions so as to ease the readability of the console listing. In general, the columns 1 to 3 of the listing are reserved for control characters and messages, input texts are printed starting from column 4 to the end of the line. The control character is intended to identify the kind of message that is printed or the type of the request to be entered (i.e. prompts).

MESSAGES AND PROMPTS	CONTROL CHARACTERS	MEANING
Prompts	S:␣	system command level
	C:␣	processor command level
	R:␣	processor request level
	I:␣	input data level
	???	break interruption
	-:␣	line continuation
Synchronous messages	>>>	start banner of a processor
	<<<	end banner of a processor
	␣␣␣	message corresponding to normal result
	␣␣*	error of severity 1
	␣**	error of severity 2
	***	error of severity 3
Asynchronous messages	⟶	messages received through the user's mailbox
Asynchronous message with a deferred reply	nn/	nn represents the system number used to identify the reply when entered
		Note: ␣ = blank

LOG-ON AND LOG-OFF PROCEDURE

The manual procedure is fully described in the 'Terminal Operation Manual'. The following is an example of a TTY-like terminal connected through a switched line with the 'controls' log-on option. IOF needs a user name (may be the terminal-id) to be cataloged in the site catalog.

ACTION	TERMINAL ACTIVITY	REF.
break signal	depress the break key	1
listener request	10.32 ID,USER/PROJECT/BILLING,APPL?TOO1, ATKINS,IOF	2
listener request	10.32 PASSWORD?	3
listener message	10.32 TOO1 ACTIVATED FØR IØF ØN FEB 28,78	4
IOF banner message	>>> 10.33 IOF 10.01	5
of today	⟶ MOT:IOF SESSION TERMINATES AT 12.00	6
ready message	S:	7

	---	
log-off request	S:QUIT;	

ACTION	TERMINAL ACTIVITY	REF.
log-off banner	<<< 11.15 ATKINS LOGGED OFF CPU 0.572 ELAPSED 79.012	8

Explanations of the above references:

1. The user may either press the break key or type the $*$ BRK BTNS command, if there is no break key which also causes an interrupt.

2. The system requests the user to enter his identification which consists of:

   ID-mandatory with a maximum of 4 characters. This is the terminal-id as declared in the network configuration.

   USER-mandatory with a maximum of 8 characters. This user name must both be unique and must exist within the site catalog.
   PROJECT-optional with 8 characters maximum. The specified project-id overrides the default project-id attached to the user.

   BILLING-optional with 8 characters maximum. Account which overrides the default one attached to the project.

   APPL-mandatory with 8 characters maximum. This is the application name. The 3 characters 'IOF' must be entered to be logged-on with the IOF facility although this parameter may not be requested if the terminal has been dedicated to IOF at network generation time.

3. PASSWORD attached to the user name, it is mandatory.
4. The listener acknowledges the user's identification and requests IOF. In other words the listener requests the scheduler to initiate a job and start a new IOF step for this new user. Note that the scheduler may deny the request if the machine has too many users.

5. The IOF log-on banner. This indicates the IOF version number (i.e. the actual state of the patches in the IOF load module).

6. and 7. First, IOF lists the messages that have been sent to the user's mailbox whilst he was logged-off. It may also display the Message of Today from the system (which has been entered by the main operator by using the MOT command) if it exists. Then, it issues the system level prompt.

8. The log-off is simply triggered by means of the QUIT command. IOF displays a log-off banner with information about the CPU time used and the total elapsed time of the session (in minutes and thousandths of a minute).

MESSAGES

The IOF user receives two types of messages at his terminal; asynchronous messages and synchronous messages. These messages are to be distinguished from

prompts. Prompts as we have seen, inform the user of the processing level of the terminal and are a request to the user to enter commands or lines of text. (We have seen that the control characters printed in the first three character positions on the display or printline tell us what type of message follows). However, messages received inform the user about the state of his program, for example, or information sent by another user. See Figure 2-1 for the relation of message types.

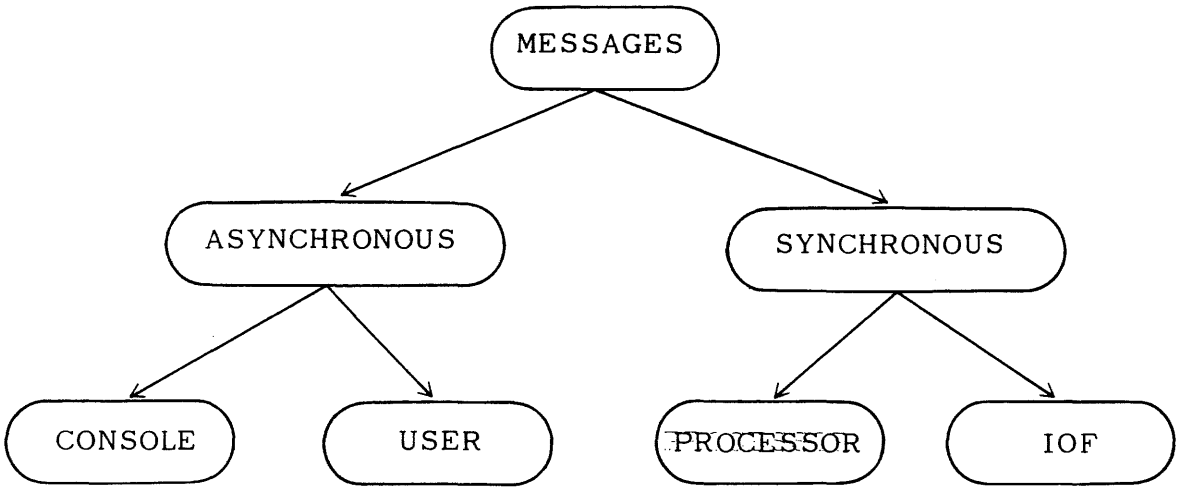


Figure 2-1 Message Types

We can see from the above figure that both asynchronous and synchronous messages are further subdivided into other types of message. The asynchronous type messages are subdivided into Console and User messages; this reflects their origin. Console messages (sometimes known as System messages) are asynchronous messages sent from GCOS. User messages are those messages which are actually sent from the user's program, for example, using the COBOL ACCEPT/DISPLAY statements.

Synchronous messages are also subdivided into other types of messages; Processor and IOF messages. As with the Asynchronous messages the names Processor and IOF reflect the origins of this type of message. The IOF messages are sent directly from IOF. The Processor messages are sent directly from the called processor, for example LIBMAINT.

Asynchronous Messages

All asynchronous messages are queued in an area called a mailbox. The messages which are stored in the mailbox are organised by user-name and not under terminal-id. As we have seen asynchronous messages can be sub-divided into two types: asynchronous messages which are queued with no reply and asynchronous messages with a deferred reply.

Asynchronous messages which are queued are displayed on the terminal or display as follows:

```
···>| QUEUED ASYNCHRONOUS MESSAGE
```

The first three characters show the message type; a queued asynchronous message with no reply.

Asynchronous messages which are queued in the mailbox with a deferred reply are represented on the terminal or display as follows:
nn/QUEUED ASYNCHRONOUS MESSAGE WITH DEFERRED REPLY
The first 3 characters show the message type; a queued asynchronous message with a deferred reply. The two characters nn represent the system number which is used to identify the reply when it is entered.
Following are some examples of asynchronous messages:

```
--->19.00 X32 IN SCANNER ABITOL P SPR=7
--->19.01 X32 STARTED SCANNER ABITOL P
```

As we have already seen all asynchronous messages are queued. Asynchronous messages are queued so that they may be displayed when the user is ready to receive them. Also we have seen that asynchronous messages may be sub-divided into two types:

- Console messages consist of those messages which are normally sent to the master operator from the GCOS System. More information may be found in the manual "Console Messages".
- User messages are either sent from another user or issued from a program. The mailbox mechanism may be used as a means of communication between two interactive users, through the JCL SEND command. Consider the following example:

  S: SEND 'HAVE A GOOD DAY',KATKINS;

  The prompt S: is followed by the JCL command SEND with the message in quotes; the message is sent to the user-name specified in the SEND command, in this case, KATKINS.

  We have also seen that asynchronous user messages can be issued from the user's program by using the COBOL ACCEPT/DISPLAY verbs.

Synchronous Messages

Synchronous messages are sent directly to the terminal; this type of message is not queued and the messages do not pass through the mailbox mechanism. Synchronous messages are either related to IOF or a Processor. A typical synchronous message from IOF at log-on would be the IOF start banner:

```
>>>18:26 IOF              10.01
```

and after logging-off

```
<<<20:12 ATKINS LOGGED OFF
<<<      CPU          0.831
<<<      ELAPSED      2.16
```

A typical synchronous message from the LIBMAINT processor would be:

```
>>> 19:02 LIBMAINT            20.02(  2)
```

and having finished with the LIBMAINT processor and issuing the QUIT command we would receive at the terminal:

```
<<<19:20
```

We can see again that the first three control characters inform us of the type of message following. Thus for synchronous messages, the message will always be preceded by three chevrons; the direction of the chevrons tell us whether we have just logged-on or

logged-off from a processor. The reader should refer to Figure 2-1 for the structure of messages and Figure 2-2 for the message handling and the mailbox mechanism.

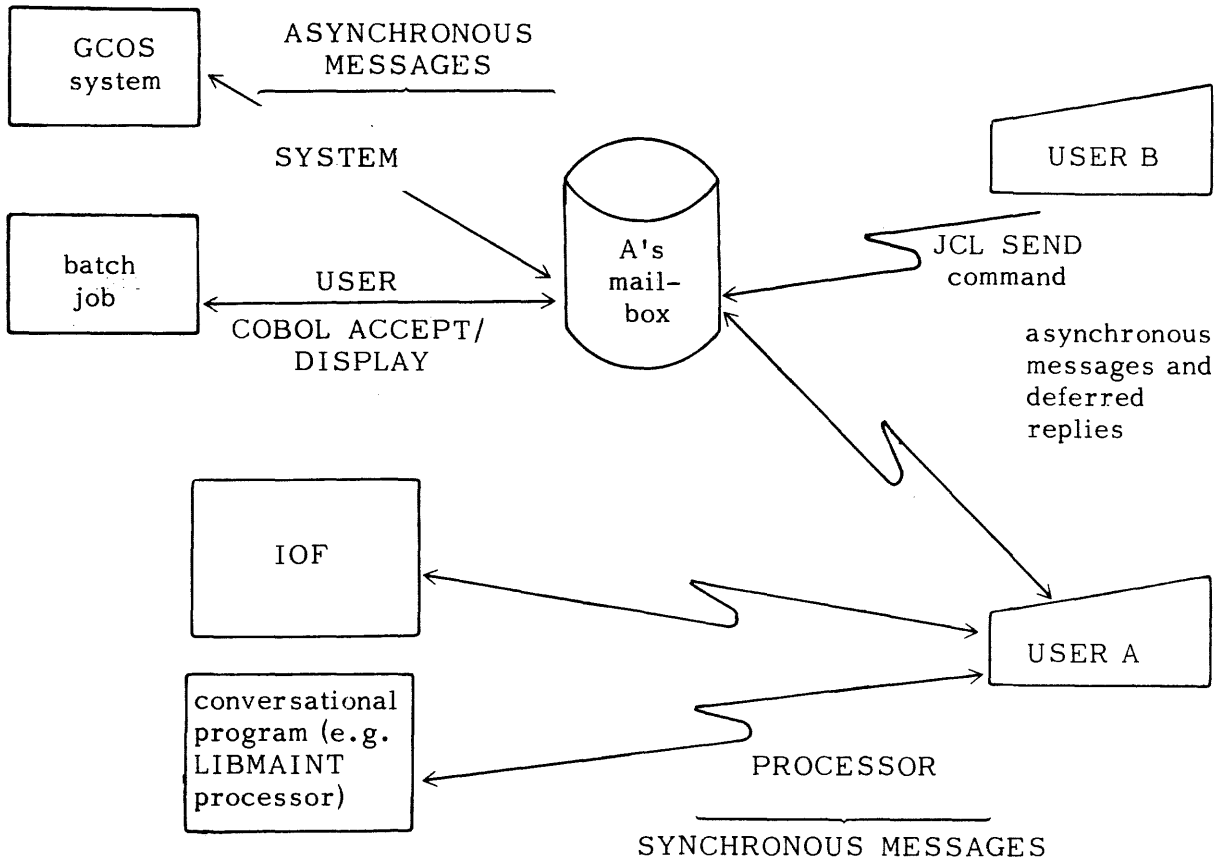


Figure 2-2 Message handling and the Mailbox mechanism.

TEXT INPUT WITH TERMINALS

Text can be entered at the terminal when the appropriate prompt is received. Input Text is entered at the terminal on one or several lines; each line being ended by the continuation character - (minus sign) followed by the transmit character, except the last line which ends with the transmit character only. Input text represents information that will be handled by the receiving program before sending back a new prompt.

The input text may contain one or several commands addressing the system or a conversational processor LIBMAINT. An input text contains a maximum of 255 characters (which represents the logical record size within source language libraries).

Note that the use of the minus sign as a continuation character is IOF-specific and has nothing to do with other continuation signs in other languages. The minus sign defines the input text continuation; other language continuation signs may be used to link successive input texts together. The input text has the same definition for all types of terminal, but the visibility may differ if a TTY-like or a VIP-like terminal is used.

TTY Type Terminals

The input text consists of one or more lines separated by minus signs (-) and new line characters (NL). The last line does not need the minus sign ending. The minus sign must precede the NL characters without any spaces or other characters between. The NL character consists of the carriage return and line feed characters. They are entered by pressing the 'new line' or 'return' keys. Consider the following example of an input text:

```
S:inputtextinputtextinputte-NL
-:inputtextinputtextinputtextinputt-NL
-:inputtextinputtextinputtextinNL
```

Since the transmission is character per character, it is simple to correct typing errors. The backslash character (\) is used to erase one or more preceding characters (Provided the ERCAP parameter was specified in the LINE statement for this particular terminal at network generation time). For example, two consecutive backslashes will erase the two immediately preceding characters. The commercial 'at' sign (@) is used to erase the whole current line. The @ character must be the last character of the line, otherwise it has no effect (so you can enter lines with @ characters in them, provided the last character is not a commercial 'at' sign (@)). Consider the following example:

```
S:inputtextinputtexxx\\tinput extNL
S:inputtextinputtextinputtextinput@NL
```

VIP Type Terminals

The input text consists of one or more lines separated by either a minus sign followed by a transmit character (TR) or by a minus sign followed by new line characters (carriage return and line feed). The last line is terminated by a transmit character without a preceding minus sign. Below we have an example of an input text using the minus sign and the transmit character:

```
S:inputtextinputtextinputtextinputtext-TR
-:inputtextinputtextinputtextinputtext-TR
-:inputtextinputtextTR
```

We should also consider the second case of using the minus sign followed by the new line character:

```
S:inputtextinputtextinputtextinputtext-NL
inputtextinputtextinputtextinputtext-NL
inputtextinputtextinputtextinputtext-NL
inputtextinputtextTR
```

Notice that since the text is transmitted only when you enter the transmit character, you do not have to wait for a prompt before entering the next line. However, the input text is limited to 255 characters and you cannot fill-up the screen before requesting the transmit. Scrolling mode is not supported and as you have just seen, work is done at line level and not at page level.

OPERATING LEVELS OF INTERACTIVE TERMINALS

The operation of a terminal is determined by the type of interaction you wish to establish with the system. It may simply be the control of a batch job which is similar

to the master operator action, in which case you are essentially interested in asynchronous messages and OCL language. On the other hand it may be an interactive activity using JCL to perform interactive executions of system processors. In this case you are entering interactive commands or data and waiting for synchronous messages and you do not want to be disturbed by asynchronous messages.

Let us see what happens after you have successfully logged-on. After log-on the terminal is at System Command level. This means you are able to request actions to be performed by the system such as call and execute interactive programs, submit batch jobs, etc. The System Command level, as we have seen in the section 'Prompts', is represented by the prompt S: . Languages available at this level are subsets of the standard Operator Control Language (OCL) and Job Control Language (JCL). This command level is either one of two modes of operation: the console mode or the interactive mode. The interactive mode is the default one at log-on in the case of an IOF common operator, but you may switch between the two modes by using the CONS/INT commands. We should also consider the levels of facilities available to the IOF user and the escape levels the user enters after issuing a QUIT command. Assuming the user has successfully logged-on and is in the interactive mode at System Command level, let us consider Figure 2-3.

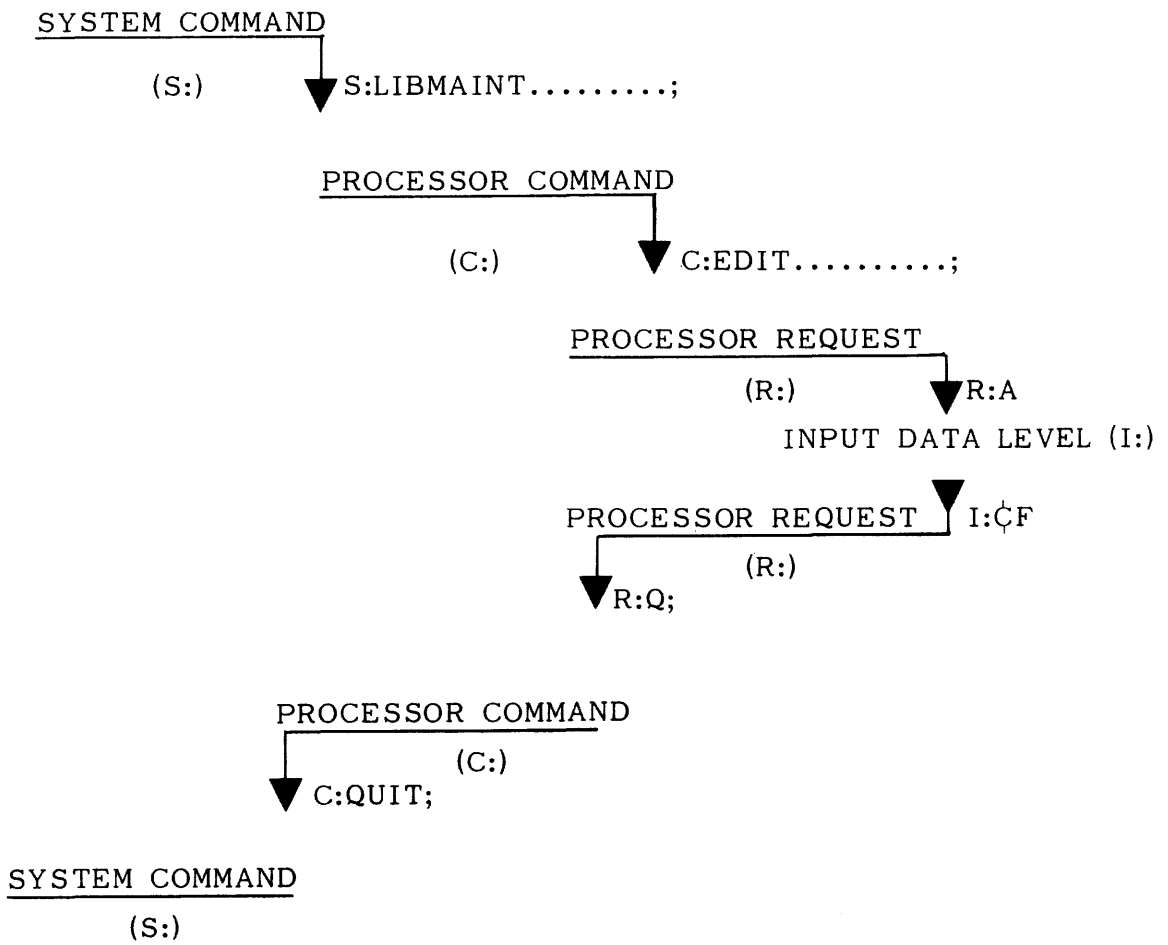


Figure 2-3 Levels of IOF Facilities and Exit Mechanisms.

The user, having entered the System Command level, can issue an OCL command

or a JCL statement. In the figure above we are calling a utility $LIBMAINT which takes us into the Processor Command level where only Library Maintenance Language is authorized. After receiving the prompt C: , which informs us that we are at Processor Command level, the EDIT command was issued which takes us into Request level. At this level only the Text Editor language is allowed and we have requested an APPEND which will take us to the Input Data level. At this level we can enter lines of source program. To exit from the Input Data level we must type F which will return us to the Processor Request level as shown above. By typing Q we can exit from the Processor Request level and return to the Processor Command level, where we can also type QUIT and return to the System Command level. It should be clear that we must pass through the Processor Request level if we enter or exit from the Input Data level. Except the input data level is also used after entering the LIBMAINT UPDATE command. Thus we must pass through the intermediary levels when we go from the highest level, which we shall call the System Command level, to the lowest level, which we shall call the Input Data level. It follows that we must pass through the intermediate levels when we exit from the lower levels. We cannot go directly from the System Command level to the Input Data level and we cannot go from the Input Data level directly to the System Command level.

THE CONSOLE MODE

In this mode the user is not solicited to enter input text; asynchronous messages that are sent to him are displayed as soon as they arrive in the mailbox. To enter an input text (which may be a deferred reply to a request message or a command), the user must issue a break signal either by pressing the break key or by typing the $*$ BRK BTNS command. The IOF system will acknowledge this by sending back the prompt S: which means the user may enter the reply or a System Command. OCL commands are handled in an asynchronous way, resulting messages are asynchronous messages which are displayed like other asynchronous messages without blocking up the terminal. However, some commands are synchronously executed within the interactive job space and all results are synchronous messages, for example the DS command.

The console mode also allows conversational communication between the logged-on user and batch jobs. All the messages and requests issued by a batch job are directed to the mailbox of the job's submitter unless a CONSOLE JCL statement overrides it. So the logged-on user will receive all the messages and requests sent by the jobs submitted from his console. Replies may be deferred or immediately displayed depending on the user's option. In the case of a deferred reply, which is the normal case, repeat the system number that has been given with the request message when entering the reply. This number is used to identify the reply with its corresponding request since several requests may be pending. Consider a COBOL program that communicates with the user through the DISPLAY/ACCEPT clauses. We want to have the program executed within a batch job and we want to control its execution. The JCL statements have been previously stored within the member CHESS of the MYLIB library. We are using a TTY type terminal and the carriage return character is not represented.

ACTION	TERMINAL ACTIVITY
System command level, console mode	S:CONS
Start CHESS job	Depress the break key S:SJ CHESS:MYLIB
Wait for job scheduling	Depress the break key S:DS CHESS

ACTION	TERMINAL ACTIVITY
Display job status (job still waiting)	bbb13.50 X30 SCH CHESS ATKINS P SPR=7 DPR=9
System message DISPLAY COBOL	--> 14.30 X30 STARTED CHESS ATKINS P --> 14.35 X30.1 CHESS:WHAT ARE YOU EXPECTING ME TO DO?
ACCEPT COBOL with a deferred reply (1)	01/14.36 X30.1 CHESS:ACCEPT WAITING ?
	Depress break key S:01 START A CHESS GAME
DISPLAY COBOL ACCEPT COBOL reply (2)	--> 14.30 X30.1 CHESS:OK,BLACK OR WHITE ? 02/14.41 X30.1 CHESS:ACCEPT WAITING ?
	Depress break key S:02 WHITE . . . . 15.00 X30.1 CHESS:ACCEPT WAITING ?
I am bored ! break signal and terminate job	Depress break key S:TJ X30
	--> 15.05 X30.1 KILLED CHESS_ATKINS P Depress break key S;INT
Return to the interactive mode	S:

THE INTERACTIVE MODE

As opposed to the console mode, a terminal in the interactive mode is, by default, ready to accept system commands. The user is advised that he can enter a new command by the prompt S: Also in the interactive mode, asynchronous messages are printed at the time a System Command Level prompt (S:) or a break prompt is printed out and just before it. They are stacked at issuing time within the user's mailbox until a system message is to be displayed. Such a mode is the normal mode of operation for an interactive user who is doing program or JCL preparations. Let us consider an example. We will take the same example as above but will describe the whole operation starting from the writing of the COBOL program, its compilation and linking and finally the storage of JCL procedures. We are using a TTY-like terminal and are at System Command level. Carriage return characters are not represented. We are using the MYLIB library to store source programs and JCL procedures. This library is supposed to be stored on a resident disk.

WRITING THE CHESS PROGRAM AND STORING IT

ACTION	TERMINAL ACTIVITY
Call LIBMAINT LIBMAINT command level	S:LIBMAINT SL LIB=MYLIB; C:EDIT;

ACTION	TERMINAL ACTIVITY
EDITOR append request	R:A
Input data request level	I:IDENTIFICATION DIVISION. I:PROGRAM-ID.CHESS. I:AUTHOR.ATKINS. . . . I:DATA DIVISION. . . . I:01 DATA1 PIC X (80). . . . I:PROCEDURE DIVISION. . . . I:DISPLAY "WHAT ARE YOU EXPECTING ME TO DO?" -:UPON CONSOLE. I:ACCEPT DATA1 FROM CONSOLE. . . . I:STOP RUN.
End of append	I:ÇF
Write the CHESS member in COBOLX	R:W(CBX)CHESS
Quit from the editor processor	R:Q
Number the source program	C:RENUMBER CHESS; C:QUIT; S:SJ LIBPRINT(CHESS,MYLIB);

If a user requires to print a library member on the line-printer he is obliged to use the LIBPRINT expression above. LIBPRINT would typically call the following JCL Statements.

```
ATTACH CATALØG1 = &CATALØG;
LIBMAINT SL  LIB =(&2,MD= &MD,DVC=&DVC)
CØMMAND = ' PRINT ' &1 ';';
```

However if the user library is catalogued in a private user catalog named PRIV.CATALØG, then the command will be:

```
S:  SJ LIBPRINT(MEMB1,MYLIB,CATALØG=PRIV.CATALØG);
```

The program is entered in the COBOLX format which is the best format to manipulate COBOL instructions at a terminal. In this format, input records to the

compiler contain only the COBOL text (from column 7 to 72 of the card format), So you just have to enter the COBOL without number (columns 1 to 6) and identification (column 72 to 80). For more details about the different formats and options see the COBOL User Guide.
Let us consider the compiling and linking of the COBOL program. We will submit a batch job as the compiler and the linker are not available in the interactive mode.

ACTION	TERMINAL ACTIVITY
JCL description of the job to be submitted	C:EDIT; R:A I:$JOB COMPL,USER=ATKINS; I:LIB SL,LIB=MYLIB; I:COBOL SOURCE=CHESS; I:LINK CHESS_LM,OUTLIB=MYLIB, I:COMMAND ='ENTRY=CHESS'; I:$ENDJOB; I:ÇF
Write the COMPL member in DATASSF form	R:W COMPL
Quit LIBMAINT	R:Q
Submit batch job	C:QUIT
Log-off from IOF	S:RUN COMPL MYLIB; S:QUIT; 14.20 ATKINS LOGGED OFF CPU : 0.572 ELAPSED 45.012

Once you have checked the compiler and linker outputs (which we can do with the interactive SCANNER function from the terminal), you can store a JCL description of the program in MYLIB and then submit the program as a job.

ACTION	TERMINAL ACTIVITY
	depress break key 17.30 ID,USER/PROJECT/BILLING,APPL? TOO1,ATKINS,IOF 17.30 PASSWORD? 17.31 TOO1 ACTIVATEDFØRIØF ØNFEB 28,78 >>> 17.32 IOF 10.01
Asynchronous messages sent by the system whilst the user was logged-off	16.35 X235 STARTED COMPL ATKINS P 16.35 X235 COMPLETED
	S:LIBMAINT SL LIB=MYLIB; C:EDIT; R:A
JCL description	I:$JOB CHESS,USER=ATKINS; I:VALUES LM_LIBRARY. I:STEP CHESS_LM,FILE=&1; I:ENDSTEP; I:$ENDJOB; I:ÇF

ACTION	TERMINAL ACTIVITY
Write the CHESS member in DATASSF form	R:W CHESS R:Q C:QUIT;
Run CHESS	S:RUN CHESS
Display Status	S:DS CHESS
Using the CØNSØLE mode	S: CØNS; --> 16.55 X321 STARTED CHESS ATKINS P 01/16.58 X321 CHESS: WHAT ARE YØU EXPECTING ME TØ DØ ?

THE BREAK SIGNAL

The break signal is entered either by depressing a break key, when there is one, or by using the BTNS command $*$BRK. The break signal normally interrupts the current processing and sets the terminal at the break interrupt level by sending the ??? prompt. The user then has three options:

1. Enter an empty command by just depressing the new line key or the transmit key in the case of a VIP terminal.
   Result: ● Under the LIBMAINT processor, the processing of the current command is stopped and the terminal is returned to the Request (R:) or Command (C:) level.
2. Enter the RESUME command (abbreviation RS).
   Result: ● The current processing is resumed at the point it was interrupted by the break.
3. Enter an OCL or a JCL command.
   Result: ● The current processing is aborted and the new system command is executed.

Examples of Break Handling:

```
                    S:LIBMAINT;
                    C:PRINT TOTO;
                      --------------
BREAK                 --------------
                    ???DS X10
                    ***TASK MAIN J=2 P=O ABØRTED BY USER
                       10.30 X10 SCH CHESS ATKINS PSPR=7 DPR=9
                    S:LIBMAINT;
                    C:EDIT;
                    R:R.ALPHA
                    R:10 $P
                      --------------
BREAK                 --------------
                    ??? NL
                    R:800,$P
                      --------------
BREAK                 --------------
                    ???RS;
                      --------------
BREAK                 --------------
                    ??? ZUT;
```

```
                *** ILLEGAL COMMAND
                ***TASK MAIN J=2 P=O ABØRTED BY USER
                S:LIBMAINT
                -:SL
BREAK           -:
                ???
BREAK           S:
                ???
                S:
```

SECTION III

INTERACTIVE COMMAND LANGUAGE

The interactive command language consists of a subset of the Operator Command Language (OCL) and the Job Control Language (JCL) with some IOF specific commands. Syntax and semantics are identical with existing languages and provide the same facilities. OCL commands are directly handled by the interactive monitor, JCL commands are translated by the standard JCL translator and generally imply the execution of a system processor (LIBMAINT). This processor must support a conversational execution as opposed to a batch execution with a command file. Only the LIBMAINT processor is provided in its conversational version, and it is restricted to the maintenance of source libraries. Interactive commands can be entered when the terminal is at the system command level. Subsequent levels correspond to the different conversational languages of the LIBMAINT processor (or other interactive programs). Consider the following example:

```
S:LIBMAINT SL,LIB=TOTO;
C:EDIT
R:A
I: ¢F
R: Q
C: QUIT;
S:
```

SYNTAX OF IOF COMMANDS

At system command level, the input text always begins with a command name followed by arguments. The command name is either a JCL statement name or an OCL command name.

$$S: \; < \left\{ \begin{matrix} \text{JCL statement} \\ \\ \text{OCL command} \end{matrix} \right\} \quad \text{name} > \; < \text{arguments} > \; [;]$$

OCL commands cannot be mixed with JCL statements within the same input text. There is only one OCL command per input text and an OCL command cannot be split into successive input texts. The continuation character must be used if the command cannot be entered within a single line. The semi-colon is optional at the end of an OCL command. Consider the following example:

```
S:SJ CHESS:MYLIB:C131:MS/M400 -(NL)
-: (PAR1,PAR2) NL
S:
```

JCL statements are completely free format; each statement must be ended with a semi-colon. There may be several statements within the same input text and one statement may be split between successive input texts. Consider the following example:

```
S:LIB SL (NL)
-:INLIB1=MYLIB (NL)
-:INLIB2=MYLIB2; (NL)
S:
```

OCL COMMANDS

The following is a list of the OCL commands that are allowed under IOF and a brief description of some specific IOF commands.
The first group control and display jobs belonging to the interactive user.

SI, SJ	submit a stream of batch jobs (see JCL RUN statement)
DS	display scheduling
HJ	hold job
RJ	release job
MJ	modify job (class,priority.....)
TJ	terminate job

The following group of commands control and display sysout deliveries (or outputs) generated by jobs submitted by the interactive user.

DO	display output
HO	hold output
RO	release output
CO	cancel output

The following facilities are provided by BTNS to display and communicate within the network and are only available with the master console

DT	display telecommunications
BT	broadcast telecommunications

The following miscellaneous group of commands are also available.

DD	display device
DTM	display time
DMM	display the memory size of the machine and the currently used memory for locked segments
CMSG	cancel all the asynchronous messages that are pending in the user's mailbox. Useful when there are too many unnecessary messages to be printed.

IOF SPECIFIC COMMANDS

These commands do not have any arguments; they are intended for terminal control.

INT	to switch to the interactive mode (terminal always waiting for an operator reply).
CONS	to switch to the console mode (terminal normally waiting for printing of asynchronous messages).
QUIT	to end an interactive session and log-off.

BREAK COMMANDS

When the system sends back the prompt ??? after a break, the following commands can be entered:

Empty command	to return to processor language level or resume current processing.
RESUME (RS)	to resume the current processing.
Other commands	to terminate the current processing and have the command executed.

A full explanation of OCL commands and the corresponding error messages can be found in the System Operation Operator Guide.

JCL STATEMENTS

The following is a list of Basic and Extended JCL commands and their abbreviations which are supported under IOF.

Basic Statements

COMMENT (C)
RELEASE (RLS)
SEND
SCAN
WRITER (WR)

Extended Statements

LIB SL
LIBMAINT (LMN)
RUN

Further information about Basic and Extended Statements can be found in the JCL Reference Manual; the Basic Statements have their full capabilities explained so we need not describe them here. Their characteristics are the same as in a batch environment apart from some exceptions which are described below.

Operation of Basic Statements

- Use of the SEND statement: this allows communication between different users through the mailbox system. Asynchronous messages are immediately displayed if the user is already logged-on or at his next log-on. Consider the following example:

```
S:  SEND 'WHAT DO YOU THINK OF A FISHING PARTY-
-:  NEXT WEEK-END ?' KATKINS
```

The message between quotes will be sent to the user named KATKINS.

Operation of Extended Statements

Extended statements are specified as there are some restrictions under IOF use.

LIB SL STATEMENT

```
LIB SL [INLIB1= {TEMP¦
                {(input-library)}]
       [INLIB2= {TEMP¦
                {(input-library)}]
       [INLIB3= {TEMP¦
                {(input-library)}] ;
```

The LIB SL statement allows input library specification to be used, according to the standard search rules, by the LIBMAINT processor. A maximum of four libraries may be declared at any one time. The last entered LIB SL statement completely overrides the preceding one; it defines the new search rules. Consider this example:

```
S: LIB SL INLIB1=S1,INLIB2=S2,INLIB3=S3;
S: LIB SL INLIB1=S2,INLIB2=S1;
```

The declared libraries in the search order are then:S2 and S1, declared at any time by entering a LIB SL statement without specifying any libraries. Consider the following example:

```
S: LIB SL;
```

No more libraries exist within the search rules.

Warning: It is possible to have a library overflow when trying to modify too many libraries. The problem can be solved by erasing all the declared libraries before entering the new LIB statement. Consider the example below:

```
S:LIB SL,INLIB1=S1,INLIB2=S2,INLIB3=S3;
S:LIB SL,INLIB=HYLIB;
*** FATAL 142 LIBS TABLE OVERFLOW
S:LIB SL;
S:LIB SL,INLIB1=MYLIB;
S:
```

LIBMAINT SL STATEMENT

```
LIBMAINT SL [{COMFILE= (sequential-input-file)  }
              {COMMAND;'command;...'            }]
            [INFILE = (sequential-input-file)
            [INDEF = (indef-parameters)]
           [{OUTFILE = (sequential-output-file)
            [OUTDEF = (outdef-parameters)]
             LIB = {TEMP | TEMP1 | TEMP2 | (output-library) } {  }]] ;
```

Restrictions: using PRTFILE is excluded and only source libraries are allowed.
The COMFILE is optional and when specified it refers only to a permanent sequential file or member of a library (input enclosures may not be used).
Further information about the Statements above can be found in the LIBRARY MAINTENANCE REFERENCE MANUAL.

RUN STATEMENT

```
RUN member (source-library)
        VALUES = (value1  ,.... keywords   =   value ,....)
        CLASS = class
        PRIORITY = priority
        HOLD
        HOLDOUT
        JOBS = ( job1  ,  job2 )
        SW = switch-values ;
```

There are no restrictions with this command.
The $RUN command provides the interactive user with the facility to submit batch jobs from his terminal. It has the same facilities as the SI/SJ OCL commands and can be used as well. All the system messages that report the execution of the submitted batch job are duplicated and sent to the user's terminal. In addition, the interactive user represents the master operator , whilst for communications the batch job is programmed to deal with the master console. The REPEAT and ROLLBACK requests are normally sent to the master operator in cases of abnormal terminations of steps using the REPEAT and JOURNAL options. These requests are sent to the interactive user provided that he is still logged on. Note that in the case of a warm restart after a crash, these

restart requests are always sent to the master operator.

For further information see the JCL Reference Manual.

LIBMAINT COMMAND LANGUAGE

The following commands are available for use with the LIBMAINT processor. More information on their use may be found in the Library Maintenance User Guide.

- CODE and DECODE commands may be used in order to provide maximum security for data stored in one or more source language units. These commands allow a user to encypher or decypher units according to a key which is specified in the command. Attempts to display an encyphered unit results in an unreadable listing.

- COMM introduces a comment that will be listed in the execution report.

- COMPARE command is used to compare two units and print out the changes made to the first one to yield the second one.

- CREATLIST command creates a unit containing a complete or selective list of the member names of a library. This unit can be later used as an "indirect name list" in commands.

- DELETE is used to suppress one or more members of a library in order to recover their space for further use.

- EDIT is used to call the Text Editor.

- EJECT causes a skip to the top of a new page in the execution report.

- ESCAPE allows a user to enter ØCL statements without having to leave LIBMAINT.

- EXEC is used to execute a series of command lines contained in a source language library unit.

- LIST is used to produce a complete or selective table of contents of a SL library or a sequential file save copy of a SL library.

- LOWER and UPPER commands respectively convert one or more source language units into lower or upper case letters.

- MOVE is to be considered with the same meaning as the equivalent COBOL verb. It is a transfer of a piece of information from an origin to a destination without altering the original information. Strictly speaking, it should therefore be considered as a copy rather than a move.

- QUIT is used to exit from interactive LIBMAINT.

. PRINT is used to display the contents of one or more members of a source language library.

. PUNCH is used to produce a card deck image of one or more source units.

. RENAME allows the user to change the name of a source unit.

. RENUMBER may be used to change the internal line numbering of one or more source units.

. SORT command sorts the lines of one or more units in ascending or descending collating sequence. The position and length of the sort key may be specified by the user.

. STATUS is used to indicate what should be done in the event of an error whilst executing a command.

. SUBMIT may be used for requesting submission of a series of JCL statements stored in a source language library unit. SUBMIT supports parameterization in a scheme similar to that of command EXEC.

. TITLE is used to introduce a title line to be printed on top of each execution report page.

. UPDATE is followed by a number of "requests", each one indicating a modification to be made on the contents of a source language library.

SECTION IV

THE TEXT EDITOR

The Text Editor is called by means of the EDIT command. This command does not specify by itself the detail of the editing operations to be performed. This is achieved by means of a number of request lines that follow the EDIT command in the command input stream. Requests are processed sequentially until a Q (quit) request is found indicating that the text editing session is terminated.

The Text Editor is a very powerful tool when its full potential is used. It is, however, designed in a modular fashion to simplify its approach for a user who wants to do rather simple things. This section will introduce the main features of the Text Editor in a progressive manner. If a user wants to be acquainted with its more elaborate potentialities, he should refer to the LIBRARY MAINTENANCE REFERENCE MANUAL, where a complete description of all Text Editor functions is given.

WORKSPACE AND ADDRESSING

Text editing requests do not operate directly on a source unit (as the UPDATE command does), but on an image of the source unit which is known as a workspace. Requests allow the user to:

- feed one or more units into the workspace
- to add lines read from the command stream into the workspace
- to modify the contents of the workspace
- to write the contents of the workspace into a unit

These requests will be introduced later on in this chapter. For the present time, we will concentrate on the layout of the workspace, irrespective of the way in which it was filled.

The workspace can be viewed as a series of continuous lines. A special case arises when the workspace contains no line at all; this case is reported as an EMPTY WORKSPACE. The only operations which are allowed on an empty workspace are to feed a unit into it and to add lines read from the input stream.

It is important to be able to refer to a particular line in the workspace. Various methods are provided for that purpose. They are designated addressing methods. There are basically three addressing methods.

Line number addressing is the simplest one; it is achieved by specifying the number of the line that one wants to address. It should be noted that the line number is generally not the rank of the line in the workspace, but its internal line number as generated by the RENUMBER or MOVE from COMFILE commands for example. If one

had created a unit with the default line numbering (NUMBER=(10,10)) and fed it into the workspace,

10 would refer to the first line

20 to the second one

100 to the tenth one

etc...

Additional means are provided to refer to the first line and the last line in the workspace. Symbol ¬ denotes the first line and would be equivalent to 10 in the preceding example. Symbol $ denotes the last line in the workspace.

Context addressing is used to refer to a line by specifying a string of characters contained in the target line. In its simplest form, the address is specified by:

/ string /

i.e. by enclosing the requested string between two slash symbols. For example:

/SECTION-3/

would refer to a line containing the string of characters "SECTION-3". The string enclosed between the two slashes is acting as a pattern to be matched on to the target line. When a match is found, the matching line is the addressed one. The pattern is known in the Text Editor as a Regular Expression.

More complex patterns (i.e. Regular Expressions) are provided in the text editor. They are introduced here with examples.

If one wants to address a line that contains a known string at its beginning, symbol ¬ is used as first character in the expression. For example, expression:

/¬SECTION-2/

will address a line that starts with string "SECTION-2"; thus line:

SECTION-234.

will be a correct target, but lines

bbbSECTION-2

or X3SECTION-3

will not.

In a similar manner, it is possible to specify that a string is to be matched at the end of a line by using the $ symbol at the end of the regular expression. For example, expression:

/ GOTO X ; $/

matches lines:

PX =33; GØTØ X;

or GOTO X;

but not line

GOTOX; Z = 4;

Other means are provided to specify that a position in the regular expression may

match any character by indicating a period character in that position. For example expression:

/A.BC/

would match lines containing strings:

AABC

ABBC

AXBC

etc...

It is also possible to specify that a character may appear optionally or as many times as required to match the target line. This is achieved by following the optional character with symbol *. Hence, expression

/INX*TO/

will match lines containing strings

INTO

INXTO

INXXTO

INXXXXTO

etc...

Note that if one wants the optional character to appear at least once in the required position, this has to be specified by stating the character twice, the second occurrence being followed by the *. Hence:

/INXX*TO/

would match all of the preceding lines except the first.

It is also possible to combine two or more of these devices to address a line. For example, the sequence of characters .* would match any occurrence of any character. Examples of regular expression follow.

Expression	Matches	Does not match
/ABCD/	xxxABCDyyy	AxBCD
/¬ABC/	ABCxyz	xABC
/ABCD$/	xyz ABCD	ABCxyz
/¬ABC$/	ABC	xABC ABCy
/AX*B/	AB AXB AXXB	AXYB
/AXX*B/	AXB AXXB AXXXB	AB AXYB
/A.*B/	AB AxB AxyzB	AC

Expression	Matches	Does not match
/A..*B/	AxB AxyB AxyztB	AB AC
/¬ABC.*DEF$/	ABCDEF ABCxy....DEF	xABCDEF ABCxyDEFz

The last addressing method is relative addressing. This is achieved by specifying an increment or a decrement to one of the two preceding forms of addresses. For example:

$-1	addresses the last but one line
¬+1	addresses the second line
30-2	addresses two lines before line number 30 (i.e. line 10 if numbered with default option)
120+6	addresses six lines after line number 120 (i.e. line 180 if numbered with default option)
/ABC/+1	addresses the line following the one containing string "ABC"
/DEF/-6	addresses six lines before line containing string "DEF"
etc...	

Before beginning to introduce the first Text Editor requests, it is necessary to introduce another concept: the notion of current line. The current line is the last line on which an operation was performed; all editor requests leave the current line pointer with a well defined value that can be used in a subsequent request. The user can refer to the current line by means of symbol .(dot). For example:

.	addresses the current line
. + 1	addresses the line following the current line
. - 6	addresses the line which lies six lines before the current line

TEXT INPUT

As explained before, there are two means by which text can be fed into an empty workspace:

- by reading the contents of the workspace from a source unit
- by entering lines into the command stream and appending them into the workspace

Reading a unit into the workspace is achieved by means of the R (Read) request, the syntax of which is:

R unit-name

Search rules apply, unless the name of the unit is explicitly prefixed by one of the

library key words.

Examples:

```
R  INLIB2 : MYUNIT
R  LIB : PROGRAM2
```

If the workspace is initially empty, its contents after the read request are the same as the contents of the read unit. If the workspace was not initially empty, the contents of the read unit are appended after the last line in the workspace. Thus, issuing a series of read requests in succession results in the physical appendage of the read units into the workspace.

Example:

```
A contains:   AX
              BX
              CX
              DX

B contains:   YY
              01
              ZZ

C contains:   X
              Y
```

If the workspace is initially empty and the following requests are entered:

```
R A
R B
R C
```

workspace contains as a result the series of lines

```
AX
BX
CX
DX
YY
01
ZZ
X
Y
```

in that order.

It is possible to specify that the appendage of the read unit be made at a place other than the last line of the workspace. This is achieved by specifying an address (as defined earlier) as the left argument of the read request. For example, if the workspace is initially empty, requests:

```
R A
/CX/ R C
```

result in a workspace containing the series of lines:

```
AX
BX
```

```
CX
X
Y
DX
```

where the workspace was first fed in with the contents of A, then the contents of unit C were appended after the line containing string "CX".

When a read request is executed, the current line pointer is set to the last line read into the workspace.

The other method for entering text into the workspace is to append text given in the command stream into the workspace. This is achieved by means of the A (Append) request whose form is:

```
A
-
-
-   literal text lines to be input
-
-
-
¢F
```

The A request is followed by the series of lines to be fed into the workspace, end of the series of input lines being indicated by the sequence of characters "¢F" at the beginning of a new line.

When the workspace is initially empty, the A request loads the workspace with the input lines. If the workspace already contains some text, the input lines are appended after the current line. For example, the series of requests:

```
R  UNIT1
A
}  text
¢F
```

will result in a workspace containing the contents of unit UNIT1 followed by the literal text given as input in the A request. This is due to the fact that the R request has positioned the current line pointer to the last line read (i.e. the last line of UNIT1).

It is also possible to append literal text at any place in the workspace. As in the R request, this is achieved by giving an address as the left argument of the A request. For example, the series of requests:

```
R UNIT1
$-2 A
}   Text 1
¢F
¬+ 1 A
}   text 2
¢F
```

will result in a workspace containing UNIT1 with Text1 appended after the second from last line (before the last but one) and with text2 appended after the second line of UNIT1.

A variation of the Append request is the I (Insert) request. When applied to an empty workspace, it behaves like the A request. When applied to a workspace containing text, appendage of the input text is made before the current line or before the addressed line. Thus, request:

¬ I

≹ text

¢F

would append literal text before the first line in the workspace.

In both cases, the current line pointer is set to the last text line appended (i.e. the last one input).

SIMPLE REQUESTS

This subsection introduces requests which are most frequently used in simple text editing situations.

One of these requests has already been introduced, it is the Q (quit) request that terminates the Text Editor session. Lines following the Q request will be interpreted as LIBMAINT commands.

The Q request by itself does not store the results of any text editing which might have been done. This has to be asked for by means of a W (write) request which writes the contents of the workspace into a specified unit of the LIB library. The format of the W request is:

W (type) [LIB:] name

The type specified in the command is the type to be given to the created unit. If the named unit already exists in the library, the request will be rejected. If the user wants to overwrite an existing unit, he must use the Z (overwrite) request which is a variation of the W request. Its format is

Z [(type)] [LIB:] name

In this form it is not necessary to specify the type to be given to the unit; when omitted, the former type of the unit will be kept.

The normal layout of a simple edit session will thus be:

EDIT;

R unit

≹ edit requests

W or Z (type) unit or new unit

Q

The W and Z requests allow the user to write only a part of the workspace into a source unit. For this function, the reader should refer to the LIBRARY MAINTENANCE REFERENCE MANUAL.

We shall now discuss the specific edit requests. These fall into two broad classes: those that modify the workspace and those that display some parts of the workspace contents without altering them.

The simplest edit request is the locate request which is used to set the current line pointer on to a desired line. Its format is simple, it consists of an address alone on the request line. Thus:

```
$
¬
100
10 + 3
/¬ ABC/ +2
etc...
```

are valid locate requests.

The locate mechanism consists in scanning the workspace from the current line pointer down to the last line until the addressed line is found. If not found in this section, the search continues from the first line down to the current line. If the search still fails, an error is reported in the form "SEARCH FAILED". Once the search is successful, the matched line is displayed and the current line pointer is set to that line.

There are two variations to the locate request. The N (No operation) request which has the form:

```
address N
```

and acts like the locate request with the difference that the matched line is not displayed when found. Example:

```
120 N
$N
/XYZ/N
```

The backward search request (<) searches for a line containing a string starting backwards from a specified address (i.e. from the address upwards to the first line, then, if needed from the last line upwards to the specified address. Its format is:

```
[address] </regular expression/
```

If the address is omitted, search starts from the current line. Thus:

```
</PARAG-2/
120</¬ ABC/
/ABC/</¬XYZ$/
```

are valid backward search requests.

All variations of the locate request set the current line pointer to the matched line

and issue a diagnostic if no match is found.

Other requests that do not alter the contents of the workspace are the print, the print with number and the print number requests. Their basic format is:

$$\left[\text{address1}\ \left[,\text{address2}\right]\right]\ \left\{\begin{matrix}P\\L\\=\end{matrix}\right\}$$

P means that the target lines should be printed without their line numbers, L requests a print with numbers and = is used for printing only the line number.

When used with no address (i.e. L, P or =), the requests means that the current line is to be printed. When used with one address, the addressed line will be printed. Examples of use are:

```
$ P
¬ +5 L
/ABCDEF/L
.-3=
```

When used with two addresses, the range of lines starting at the line addressed by the first address and ending at the line addressed by the second address inclusive is printed. In particular, the following request:

```
¬,$L
```

will print the whole contents of the workspace. Examples of requests are:

```
$-E, $ P
¬,¬+2 L
/ABCD/ , /ABCD/ +4 P
100, 150 L
100, 100+10 P
etc...
```

Note that 100 + 10 does not mean 110 but 10 lines after line number 100.

Requests that alter the contents of the workspace are the C (Change), D (Delete) and S (Substitute) requests.

The C request has a behaviour similar to that of the Append request. Its format is:

$$\left[\text{address1}\ \left[,\text{address2}\right]\right]\ C$$

$$\left\{\ \text{literal text}\right.$$

$$¢F$$

This means that the range of lines defined by the two addresses is replaced by the literal text lines that follow the request up to the "¢F" string. If the second address is omitted, the changed range is limited to one line, namely the one addressed by address 1. If both addresses are omitted the current line is changed. After a C request, the current line pointer is set to the last changed (input) line. It should be noted that a range of lines may be replaced by any number of lines, the number may be smaller, greater

or equal to the number of lines in the changed range of lines.

The D request, deletes a range of lines. Its format is:

[address1 [,address2]] D

When one or both addresses are omitted, the same rules as in the C request apply. In particular, a D alone will delete the current line. The current line pointer is set to the line immediately following the last deleted one. Examples of delete requests follow:

/ABC/ , /ABC/ + 10 D

delete from line containing string "ABC" down to 10 lines after this line.

¬,$D

delete all workspace. The workspace becomes empty.

$D

delete last line.

120 D

delete line number 120.

D

delete the current line.

The Substitute request is slightly more complex. It is used to replace all occurrences of a given string in a specified range of lines by a new string. The general format of the S request is:

[address 1 [,address2]] S/regular expression/string/

The range of lines where the substitution applies is determined by address 1 and address 2 with the same conventions as for the C or D requests. All strings that are matched by the regular expression are replaced by the specified string. Examples of use follow.

Initial string	Requests	Resulting string
THE BROWN SOX	S/SOX/FOX/	THE BROWN FOX
THE BROWN FOX	S/¬THE/A/	A BROWN FOX
A BROWN FOX	S/A/TWO/	
	S/$/ES/	TWO BROWN FOXES
TWO BROWN FOXES	S/FOXES/DOGS/	TWO BROWN DOGS
TWO BROWN DOGS	S/BROWN//	TWO DOGS

etc...

(Additional functions of the S request will be found in the LIBRARY MAINTENANCE REFERENCE MANUAL). At the end of the S request, the current line pointer is set to the last line in the addressed range.

If one wants to operate a substitution over the complete workspace, one should write

¬,$S/regular expression/string/

If no matching string is found a diagnostic is issued by the Text Editor.

MORE ELABORATE REQUESTS

Following is the description of requests that are less frequently used than the preceding one.

It has been seen that the P,L,D and = requests can apply to a range of lines by specifying the beginning and end addresses of the range. The requests apply to the whole range which implies that they can be used only if the target lines are contiguous. The Global requests (GP,GL,GD,G=) generalise the scope of the P,L,D and = requests to apply to non contiguous lines within a given range of addresses. The format of the Global requests is:

$$\left[\text{address1}\ \left[,\text{address2}\right]\right] \text{G} \begin{Bmatrix} \text{P} \\ \text{L} \\ \text{D} \\ = \end{Bmatrix} \text{/regular expression/}$$

This means that the P,L,D or = request is to be applied to all lines in the specified range that are matched by the specified regular expression. If no address is specified, the whole workspace is assumed (¬ ,$). For example, request

GD/¬ABC/

would delete all lines in the workspace that start with string "ABC". The request GL or GP is particularly useful to list all lines of a workspace that meet a certain criterion (i.e. that contain a specified string or match a given pattern).

A variation of the Global request is the V (Exclude) request that applies the P,L,D or = request to all lines that do not meet a specified criterion. For example,

VD/¬ABC/

would delete all lines in the workspace that do not start with string "ABC".

Other more elaborate requests may be used:

- to count the number of lines that contain a given string (≠request);
- to break a line into two or more pieces (% request);
- to merge two or more lines into a single line (& request).

For these requests, the reader should refer to the LIBRARY MAINTENANCE REFERENCE MANUAL.

USING AUXILIARY WORKSPACES

Up to this point in the discussion of the Text Editor, we have assumed the existence of only one workspace. This workspace, known as the current workspace is the object of all requests: i.e. editor requests, displays or modifications of the contents of this workspace.

In fact, Text Editor supports up to six workspaces. However, at any moment, only one workspace is known as the current workspace; other workspaces are known as auxiliary workspaces. Workspaces are named 0,1,2,3,4 and 5 respectively. When one enters the EDIT command, workspace 0 is created and designated as the current workspace. To create a new auxiliary workspace, it is sufficient to mention its name in one of the workspace requests detailed below. When first mentioned, the auxiliary

workspace is initially empty.

To change the current workspace, the user must issue a B request (Change Base or Workspace). The current workspace becomes an auxiliary workspace and is left as it is, the designated auxiliary workspace becomes the current workspace. All subsequent edit requests will apply to this new current workspace until another B request is encountered. The format of the B request is:

B(x) with x=0,1,2,3,4 or 5.

If in doubt about which is the current workspace, the user can issue an X request which displays the status of all known workspaces. The current workspace is identified by an arrow and the number of lines in each workspace is displayed. The format of the X request is:

X

and a typical output would be:

```
*WORKSPACE (0)          120
*WORKSPACE (2) ——→      140
*WORKSPACE (5)          220
```

Three requests allow transfer of text from the current workspace into a designated auxiliary workspace.

The copy (K) request copies lines from the current workspace into an auxiliary workspace. The current workspace is left unchanged, the previous contents of the auxiliary workspace are lost.

$$\left[\text{address 1}\ \left[,\text{address 2}\right]\right]\ \text{K(x)}$$

The move (M) request works like the K request with the difference that the copied lines are deleted from the current workspace

$$\left[\text{address 1}\ \left[,\text{address 2}\right]\right]\ \text{M(x)}$$

The file output (F) request specifies that all results from a print (P, L,=,VP,VL,V=,GP,GL,G=,etc...) request be appended at the end of a specified workspace instead of being printed on the execution report. The effect of the F request is reversed by an E request.

F(x)

E

A typical example of auxiliary workspace use follows.

Example: Create a unit containing all lines of a source unit that contain a GOTO or a PERFORM statement.

EDIT;	enter editor
R UNIT	feed the source unit (in workspace 0)
F(1)	file output in workspace 1
GP/GOTO/ GP/PERFORM/	printed lines are filed into workspace 1
E	end file output
B(1)	Change current workspace to 1
W(COB)RESULT	Write result
Q	Terminate

WRITING EDITOR PROGRAMS

One of the important capacities of the Text Editor is its ability to read requests from an auxiliary workspace. This is achieved by the ¢B(x) sequence of characters. Each time that this sequence is found, it is processed exactly as if the current contents of workspace x had been inserted at that point. This means, in other words, that the request stream is provisionally redirected towards the designated workspace. Return to the original request stream is made when the end of the workspace is reached or when a request executed from the workspace results in a SEARCH FAILED diagnostic.

An important use of this feature is for inserting the same pattern of lines at different places in a unit. It is processed as follows:

1. feed the string (pattern of line) into an auxiliary workspace (say 2):

   B(2)

   A

   ≷ lines

   ¢F

2. Return to the former workspace

   B(0)

3. At each place where the string should be inserted requests should be issued

   A
   ¢B(2)¢F

   This means that once the A request is read in, subsequent lines will be read from workspace 2; when workspace 2 is exhausted, ¢F is read from the request stream. This construction achieves the desired result.

The above construction may be used to write complete editing programs into a workspace and to call them by means of the ¢B(x) mechanism. Additional requests allow for branching, looping and testing. These are:

:x
define a label at this line

)Lx
go to label x

)$\overset{+}{-}$n
go n lines forwards or backwards

*/expression/request
execute request if current line contains expression

address? request
execute request if current line pointer is at specified address

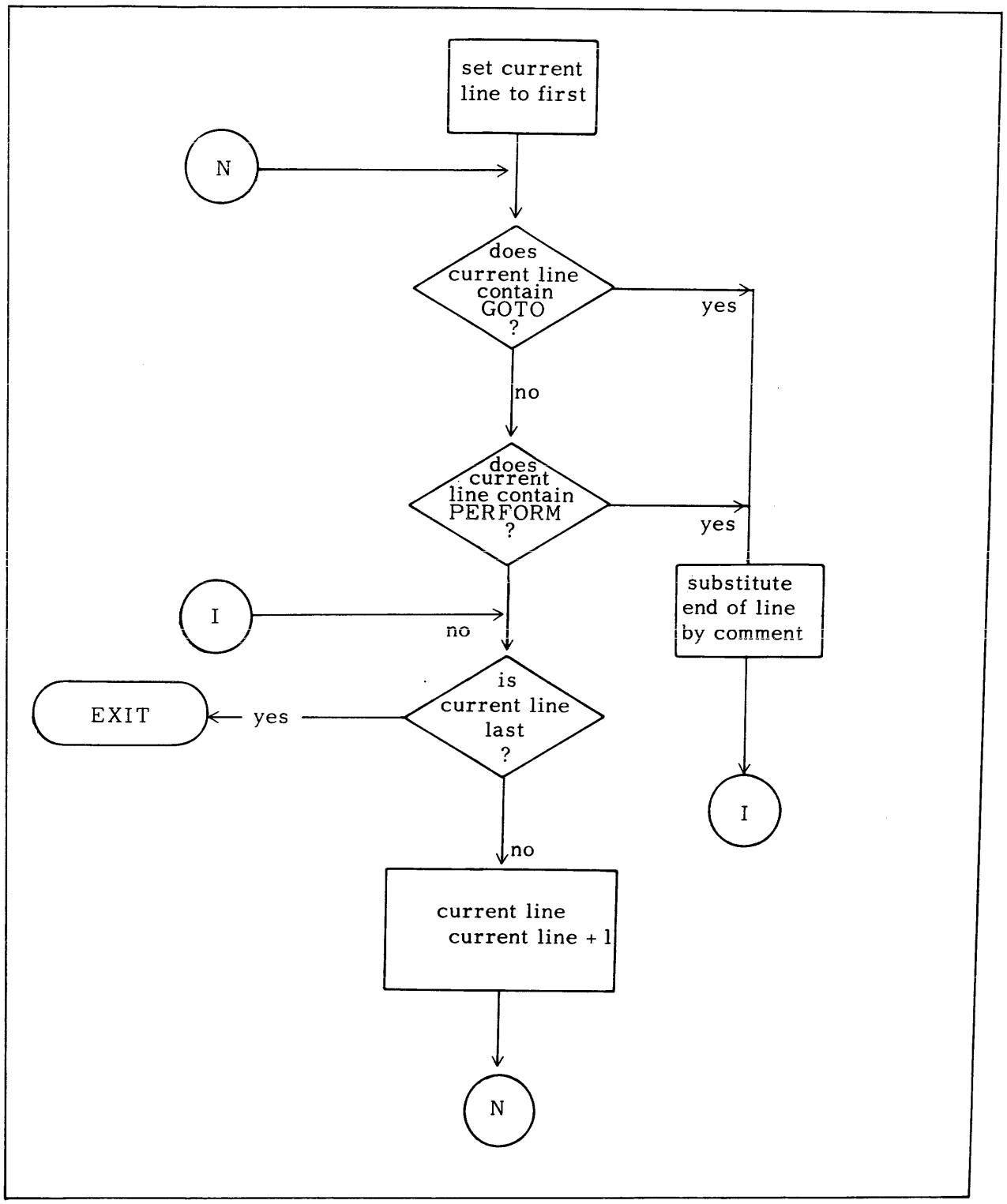
Detailed information on the use of these requests will be found in the LIBRARY MAINTENANCE REFERENCE MANUAL. An example of an editor program will illustrate some of these programming facilities.

Example:

Write an editing program that will scan all lines of a workspace and introduce a comment at the end of each line cor taining GOTO or PERFORM.

The following sequence may be introduced into a workspace and called by a ¢B(x) sequence when requested.

Flowchart of the operations to be performed is given below:



An Editor program implementing this algorithm is given on the following page.

¬N	position on first line (No operation)
: N	define label N
*/GOTO/> LR	if line contains GOTO then go to R
*PERFORM/> LR	if line contains PERFORM then go to R
:I	define label I
$? >LE	if last line go to E
.+1 N	move one line forward (No Operation)
>LN	go to label N
: R	define label R
S/$/comment/	add comment at end of line
>LI	go to label I
: E	define label E for exit.

CONTROL REQUESTS AND ESCAPE SEQUENCE

Additional requests in the editor allow the user to control the type of output that he wants from the editor and the action to be taken in case of errors. These requests are known as control requests and are introduced by letter Y.

Another type of editor control is provided by an escape sequence that acts as a kind of macroprocessing on the editor requests. Escape sequences are introduced by character ¢. Two of these have already been introduced:

¢F to denote end of A,C or I requests

¢B(x) to invoke the contents of a workspace.

Both the control requests and escape sequences are discussed in detail in the LIBRARY MAINTENANCE REFERENCE MANUAL.

SECTION V

INTERACTIVE LIBRARY MAINTENANCE

Interactive LIBRARY MAINTENANCE is basically the same product as batch LIBRARY MAINTENANCE. This means, in particular, that functions offered, syntax of commands and behaviour of the system are the same in batch and interactive modes.

This chapter will specifically concentrate on the aspects of interactive LIBMAINT. Differences between batch and interactive modes will be listed and indications on how to enter in and to react to interactive LIBMAINT will be given. Further details of the batch and interactive modes of LIBMAINT may be found in the LIBMAINT User Guide and the LIBRARY MAINTENANCE Reference Manual.

ENTERING INTERACTIVE LIBRARY MAINTENANCE

Interactive LIBRARY MAINTENANCE operates under Interactive Operation Facility (IOF) and is fully integrated to that system. It is assumed that the reader is familiar with the basic concepts of IOF and has logged-on.

From most interactive processors' standpoint, the user's console is viewed as a job stream: i.e. a succession of JCL statements, of commands and of data subordinated to those statements and commands. To enter LIBMAINT, it will thus be necessary to enter suitable JCL statements to indicate the various entities INLIBs, LIB, INFILE, OUTFILE on which LIBMAINT is to operate. This is achieved, like in batch, by the LIB and LIBMAINT statements.

Interactive LIB statement has the same purpose and syntax as in batch mode. LIBMAINT statement also has the same form, the only difference is that the key word COMFILE may be omitted. This means that commands, requests and data can be input from the user's console rather than from an input enclosure or a file. Both LIB and LIBMAINT statements may be entered at any time when IOF is at "System Level", i.e. has issued the prompt S: and is expecting an input. Following is an example of how to enter interactive LIBMAINT.

```
S:        LIB SL,
_:        INLIB1 = (MYLIN1,DEVCLASS=MS/M400,MEDIA=C100);
S :       LIBMAINT  SL,LIN=TEMP;
>>> 12:07 LIBMAINT  20.01  (3)
C :
```

In this example, text which is input by the user is underlined and system output is not.

First the user entered a LIB statement at S: level; as the statement was not terminated on the first line, system requested for a continuation by issuing the prompt -: . On the third line, the LIBMAINT statement is issued without a COMFILE; this means that commands will be interactively accepted from the console.

At this point, the user has entered interactive LIBMAINT. On the following line, LIBMAINT identifies itself, then requests a command by issuing the prompt C:. The user is now free to enter any valid LIBMAINT command. If the command does not terminate on the line (end with a semi-colon), LIBMAINT will request for a continuation by issuing the prompt -:

Example:

```
C:  MOVE  INLIB1:MYUNIT,
-:  NUMBER,
-:  REPLACE;
C:
```

Once a command is completely entered it is immediately executed and any possible results are printed on the console, followed by a prompt (C:) requesting a new command.

If a faulty statement is entered, the whole statement is reprinted with a diagnostic and a pointer to the faulting item. Then a new command is requested.

Example:

```
C:  MOVE  XNLIB1 : MYUNIT,
-:  REPLACE;
    MOVE XNLIB1:MYUNIT,REPLACE;
         ?
***  UNKNOWN LIBRARY KEY WORD
C:
```

From the above examples, it can be seen that the first three columns of the console output are used to prompt the expected users reaction. This mechanism is extended to other levels of LIBMAINT. For example, if at the C: level one enters the EDIT; command, the next thing that LIBMAINT is expecting is a request. This is notified by prompting characters R: in the prompt area. If the user then enters an input request, following lines are expected to be input data, this is notified by prompt I:, until exiting from input mode, where prompt R: will be reissued. If the user issues a Q request at request level (R:), LIBMAINT will return to command level (C:).

Example:

```
C:  EDIT;          <------  invoke editor
R:  A              <------  append request
I:  THIS IX        }
I:  MY TEXT        }  appended text
I:  ¢F             <------  exit input mode
R:  S/IX/IS/P      <------  request to substitute and print
    THIS IS        <------  result of print
R:  Q              <------  quit editor
C:                 <------  return to command level
```

From this example, it can be seen that results are indented three spaces in order not to appear in the prompt zone. This is done to achieve clarity in the console output.

A user might also wish to execute from a console a series of commands stored in a file. This can be achieved by specifying the key word COMFILE in the LIBMAINT statement. LIBMAINT will then read commands from the specified file, echo them on the console, execute them and print possible results unit the file is exhausted.

RESTRICTIONS OF USE

There are only two restrictions of use of LIBMAINT commands in interactive mode: Global EDIT and UPDATE with star convention or name lists may not be typed on the console. This would result in such a poor operability that it has been decided to forbid this construction. It is however possible to enter these statements into a source unit and to EXEC it from the console.

ADDITIONAL COMMANDS

Two commands are introduced in LIBMAINT to cater for the specific requirements of the interactive users:

command QUIT is used to exit from interactive LIBMAINT, it plays a role similar to the end of the COMFILE file in batch mode. Its syntax is very simple:

QUIT;

The command ESCAPE allows a user to enter Operator Command Language (OCL) statements without having to leave LIBMAINT. All OCL commands which are authorized at S: level may be coded in an ESCAPE command. Syntax of ESCAPE is:

ESCAPE OCL-statement;

Examples:

```
ESCAPE DS X137;
ESCAPE TJ X136;
ESCAPE SI MYJOB:MYLIB:MS/M400:C100;
```

TREATMENT OF BREAKS

One of the important features of an interactive system is the ability for its users to imperatively interrupt a process. Interrupts are notified by means of Breaks. A Break is notified as follows:

by depressing the "Break" or "Interrupt" key of the console for those consoles which have such a key

or by entering sequence $*$BRK on a new line.

When a Break is entered while LIBMAINT is processing, the system issues the characters ??? in the prompt area and expects a reply from the user. Replies fall in three categories

- RESUME; or RS; may be entered to request the continuation of the current processing at the point where it was interrupted by the Break.
- any JCL, OCL or IOF command causes the termination of LIBMAINT and the execution of the specified command.
- a carriage return (blank line) causes an interrupt of the current processing. If LIBMAINT was executing a command, the command is aborted and a new command is requested. If LIBMAINT was executing an EDIT request, the request is aborted and a new request line is asked for.

As a general rule, all precautions have been taken in order to minimize confusion that might result from use of Breaks in critical phases of the processing. This might cause a very small delay in the response of LIBMAINT to a break, because, in some circumstances, some critical action must be terminated before aborting execution of a command or a request in order to leave the object libraries and files clean.

If a Break is issued while LIBMAINT is executing a command from a COMFILE, the current command is aborted and the LIBMAINT session terminated.

If a Break is issued while LIBMAINT is executing commands through an EXEC command, the current command is aborted, EXEC is terminated and a request is made for the next command from the console.

Example

As explained above, functions offered to an interactive user of LIBMAINT are the

same as those offered to a batch user. Let us consider an example of using interactive LIBMAINT to prepare a job to be submitted from the console (Conversational Remote Job Entry).

```
S:   LIBMAINT  SL, LIB=USER.SLLIB ;          enter LIBMAINT
>>>  13:36 LIBMAINT  20.01  (3)
C:   EDIT;                                   invoke editor
R:   A                                       append
I:   $JOB  C,USER=D,ACCOUNT=E;
I:   LIB  SL,LIB=USER.MYLIB;                 job description
I:   COBOL  SOURCE=MYPROG;                   job description
I:   $ENDJOB                                 job description
I:   ¢F
R:   W (JCL) COMPL                           write in COMPL
R:   Q                                       leave editor
C:   QUIT;                                   terminate LIBMAINT
<<<  13:37
S:   RUN COMPL,USER.SLLIB;                   request execution of COMPL
```

ERROR HANDLING

In interactive mode, behaviour in the case of errors is different than for batch mode. When the user enters commands from a console, he is able to immediately react to an error by retyping the failing line or taking any suitable corrective action. Termination of LIBMAINT in the case of a SEVERE ERROR is therefore not desirable. The STATUS command is also of no interest in this context, it is therefore ignored.

However, when a user triggers the execution of a stored sequence of commands by means of the COMFILE parameter in the LIBMAINT JCL statement or by entering an EXEC command on the console, error control becomes a necessity. In these cases, LIBMAINT behaves as in batch mode:

- if commands are read from a COMFILE and a SEVERE ERROR occurs, the LIBMAINT session terminates unless a STATUS command specifies otherwise
- if commands are executed from a subfile triggered by an EXEC command entered on the console and a SEVERE ERROR occurs, execution of the subfile is discarded and control returns to the console unless a STATUS command in the subfile specifies otherwise.

LIBRARY MAINTENANCE OUTPUT REPORT

In interactive mode, the LIBMAINT report differs from the batch mode report. The first line of the session is the standard interactive banner >>> . The following lines are printed in accordance with the IOF standard rules. The prompts are

input prompts
- C: for enter a command
- R: for enter a request
- I: for enter input
- -: for enter continuation

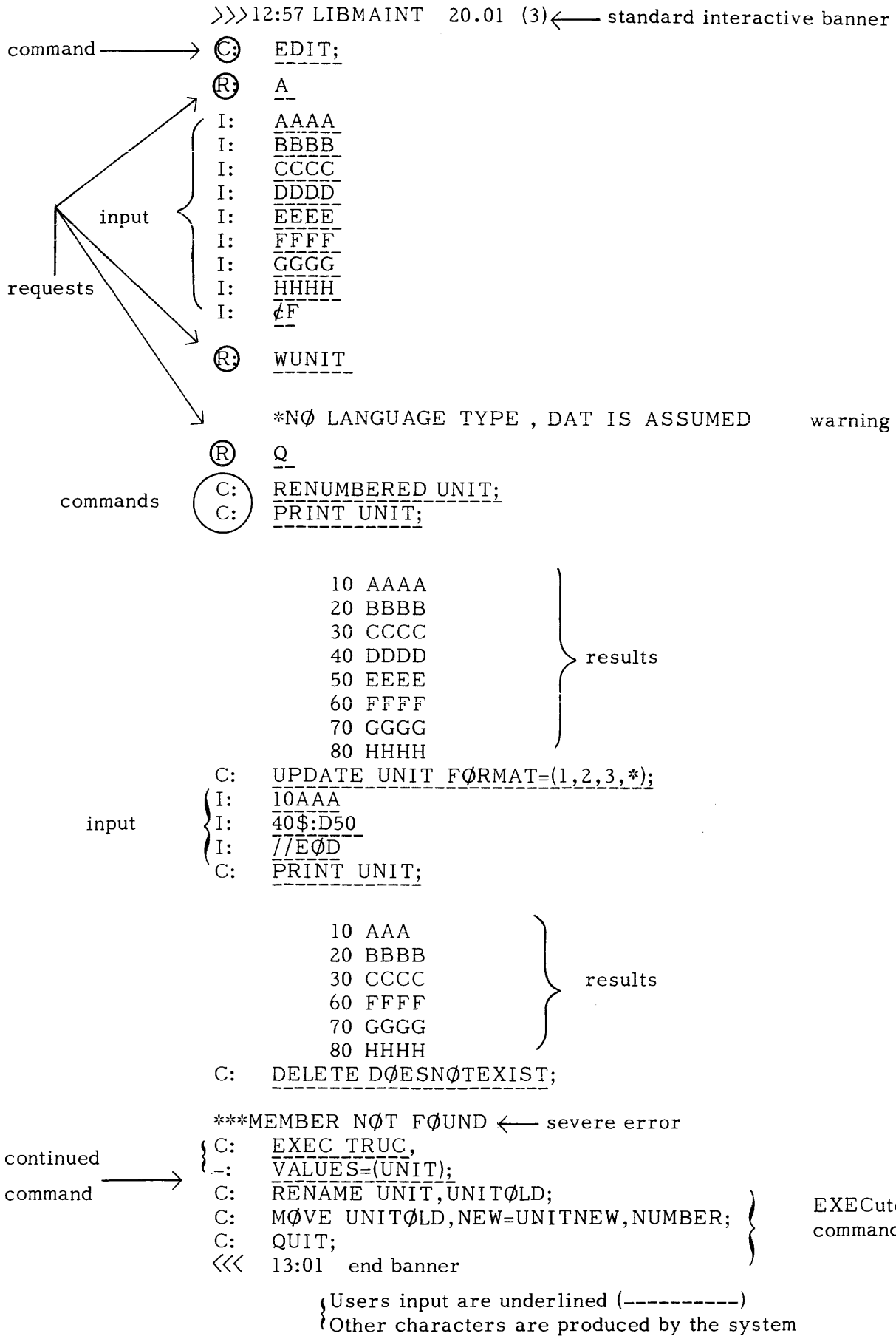
output prompts
- blank for result
- * for warning
- *** FOR severe error

Executed commands or commands entered through a COMFILE are listed as if they had been entered on the console. They are therefore preceded with prompts C: ,R: and I:.

The last line of the session is the termination banner <<< which indicates the current time of the day.

In interactive mode, identification of handled units is not printed except with the LIST command. When a command is applied to more than one unit (star convention, name list, etc...) the name of each member is printed before being processed but its identification is not printed.

A commented example of the output of a sample interactive LIBMAINT session is given in the following page.


>>>12:57 LIBMAINT 20.01 (3)
standard interactive banner
command
C: EDIT;
R: A
input
I: AAAA
I: BBBB
I: CCCC
I: DDDD
I: EEEE
I: FFFF
I: GGGG
I: HHHH
I: ¢F
requests
R: WUNIT
*NØ LANGUAGE TYPE , DAT IS ASSUMED
warning
R: Q
commands
C: RENUMBERED UNIT;
C: PRINT UNIT;
10 AAAA
20 BBBB
30 CCCC
40 DDDD
50 EEEE
60 FFFF
70 GGGG
80 HHHH
results
C: UPDATE UNIT FØRMAT=(1,2,3,*);
input
I: 10AAA
I: 40$:D50
I: //EØD
C: PRINT UNIT;
10 AAA
20 BBBB
30 CCCC
60 FFFF
70 GGGG
80 HHHH
results
C: DELETE DØESNØTEXIST;
***MEMBER NØT FØUND
severe error
continued
command
C: EXEC TRUC,
-: VALUES=(UNIT);
C: RENAME UNIT,UNITØLD;
C: MØVE UNITØLD,NEW=UNITNEW,NUMBER;
C: QUIT;
EXECuted commands
<<< 13:01 end banner
Users input are underlined (----------)
Other characters are produced by the system

SECTION VI

IOF OPERATION FROM A SYSTEM VIEWPOINT

This section describes the IOF as seen by the master operator and the system manager. Different points will be discussed, particularly the control of IOF operation, the management of system resources, the billing and the interfaces with the main operator.

IOF GENERATION

IOF uses the standard VCAM access method of GCOS-64. So the different actions necessary to initiate VCAM must be performed before starting IOF. The terminal network first has to be generated by using the Communication Network Configurator (CNC); terminals may be declared general purpose and so used can work with a TDS application, a COBOL application using MCS or IOF itself. The IØF facility is specified by the key word 'IOF' in the DCTAP command (NDL) at CNC generation time. Terminals may also be dedicated to IOF (log-on automatic under IOF), the log-on procedure must be declared as a manual log-on with controls. (The CNC is fully documented in the Communications Processing Facility Manual).

IOF cannot work without the site catalog, so the catalog must be present in the system and the interactive user names must be catalogued with a given password for each one. (The 'System Management Guide' provides a good description, see also Catalog Management).

SCHEDULING OF IOF USERS

The scheduler provides the system manager with tools to control the IOF operation concurrently with batch processing. It particularly allows the system operator to start or terminate the whole IOF operation and to set a maximum limit on the number of logged-on users. It also automatically controls the use of different resources in order to prevent a system overload due to IOF operation.

There is one interactive job being executed per logged-on user. The launching of an interactive job is performed when a successful log-on has been completed. Activation is performed only if a certain number of conditions are satisfied. Otherwise it is denied and the listener notifies the user that he cannot log-on because the system is overloaded. These conditions are related to the machine's current load, involving the IOF maximum load, the system multiprogramming limit, memory use and some system resources.

System Resources

The system resources concerned are:

- IOF maximum load: this defines the upper limit of the maximum number of users that can be logged-on at the same time. It is configurable on site with a default value of 10.
- System multiprogramming limit: this includes batch jobs, interactive jobs, file transfer jobs and service jobs (like the JCL translator job, the output writer job, the BTNS job). This number is configurable on site with a default value of 15.
- The Scheduling Class state and load: the 'Q' scheduling class is normally given to all interactive jobs. So the class state (started/terminated) is a way to start or terminate the IOF operation. Also the class maximum load is an additional way to limit the maximum number of logged-on users (from 0 up to the IOF maximum load). So if the maximum load is exceeded or if the class is terminated, the log-on is denied. These class attributes are initialized at site configuration time but they can be modified by using the OCL commands related to the scheduling classes (MC/TC/SC).
- Memory use: memory is reserved for each interactive job from their respective beginnings (log-on) to their ends (log-offs). Such reservations allow the interactive session to be processed normally without memory problems. Thus if the memory is overloaded at scheduling time the activation is denied.
- System resources: this mainly concerns the maximum number of tasks, which introduces a limit that has to be checked at scheduling time so that the resources are reserved for the whole interactive user session.

Activation Process

Once the scheduler agrees to launch an interactive job, that job is initiated and the H_IOF step is loaded and started. If the load module has been previously preinitialized, initiation is straightforward. Otherwise, loading may be unsuccessful because of a lack of backing store space. In such a case the interactive job will be abnormally terminated and the user will have to disconnect and try to log-on again. The interactive job is given the following description, (which is normally provided by the $JOB JCL statement).

Job identification:	"IOF"
Project:	project of the logged-on user.
Billing:	cost-centre of the logged-on user.
Scheduling class:	Q

Note that the user name is given at log-on time and the project and account are retrieved by default from the site catalog when not specified at log-on time.

Execution Priority of IOF Users

Interactive jobs are given the execution priority associated with the Q schedule class. This execution priority is initialized at site configuration time and can be modified by using the MC command. The execution priority is the priority of CPU allocation. So by giving a certain execution priority to the Q scheduling class, we define the priority of IOF users relative to competitive batch jobs. Default value = 4.

MASTER OPERATOR VIEW OF IOF

BTNS Functions

All the facilities provided by BTNS are relevant to IOF users. BTNS in particular keeps a record of the different log-ons and log-offs to IOF. Also, the main operator is provided with some commands that allow him to display a list of currently logged-on users, to send messages to logged-on users and to terminate interactive user sessions. All these facilities are documented in the Systems Operation Operator Guide.

Useful OCL Functions

In addition to the standard BTNS facilities, the IOF system provides the main operator with some further commands:

- Display of interactive jobs:
  DS IOF will display all the interactive jobs by giving their ron, their execution priority and their CPU usage.
  DS IOF (user-name) the same as above, but specifically for the indicated user.
- Modify the execution priority of an interactive job:
  MJ ron ** <execution-priority>
  The default execution priority of an interactive job is the priority attached to the Q scheduling class (which can be modified with the MC command). The MJ command allows the operator to modify the default priority for a given interactiv user's job.
- Start and terminate the IOF operation: this is simply done by starting or terminating the Q scheduling class with the SC/TC commands. When the Q class is terminated, any new log-on attempts are denied.
- Modify the authorized maximum number of logged-on users: this is done by modifying the maximum load of the Q class with the MC command. If the new number is less than the current number, any new log-on attempts are denied.

MANAGEMENT OF INTERACTIVE JOBS

The following refers to the standard functions of Job Management in the GCOS

system.

Resource Management

The initiation and termination of interactive job steps are handled in the standard way. However there is no queuing in the case of a conflict at resource allocation time. The requested step initiation will be aborted and the terminal will be returned at system command level with an appropriate message. This message normally gives sufficient information about the resource which is unavailable. Nevertheless, the interactive job may wait for media mounting until the operator has mounted all the requested media (disk-volumes or tapes). If a wait for media mounting occurs, a message informs the user that he will have to wait.

Job Reporting

The Job Occurrence Report for an interactive job is not normally printed either at the terminal or at the central system. It is, however, printed in the case of an abnormal termination of the H_IOF step. Sysout deliveries generated by the interactive job (including dumps) are normally printed out at the end of a job; the time of the printout depends on the user's options. By default delivery is made at the end of a job, so all Sysout deliveries generated by the interactive job are printed at the end of the session. That is, unless a user has specified a printout at the end of a step by using the SYSOUT JCL statement.

Accounting

IOF activity is fully recorded within the standard billing mechanism file. Each interactive user session is described by its job description, which consists of a job record and one step record per program. Records are identified by the characteristics of the interactive job as defined in the sub-section 'Activation Process' (JOBID= 'IOF', USER-NAME = name of the logged-on user, PROJECT=user's project, BILLING = billing identificaion). Details of the billing mechanism records may be found in the System Management Guide.

Warm Restart

Interactive jobs are not restartable at either step level or job level after a crash.

SECURITY AND ADMINISTRATION FUNCTIONS

Definitions

Administrative functions refer to the control of user access to the system together with its different privileges. Such functions are the responsibility of an IOF

administrator. The registration of users within the system and priveleges related to IOF operation are logically part of an integrated privacy protection system. Such a system is implemented within the GCOS site catalog.

Description Of User Registration

A user is identified within the system by a user identifier of up to 8 characters. Such a user-name must be unique; all the users known to the system must have different user-names. A user-name is always attached to a project identifier. A project-id has different user-names associated with it. See figure 5-1. A project-id is attached to a billing identifier of up to 8 characters; a billing-id has several project-ids associated with it.

Such a structure, which can be represented by a tree structure is stored in the catalog system. Generally a user has a default project and a project has a default billing mechanism associated with it, but a user may exist under different projects and billing mechanisms. The project is related to the protection mechanism in general. Communication between the user and the system, and between one user and another themselves is supported via a mailbox system. A mailbox is attached to each user registered within the site catalog.

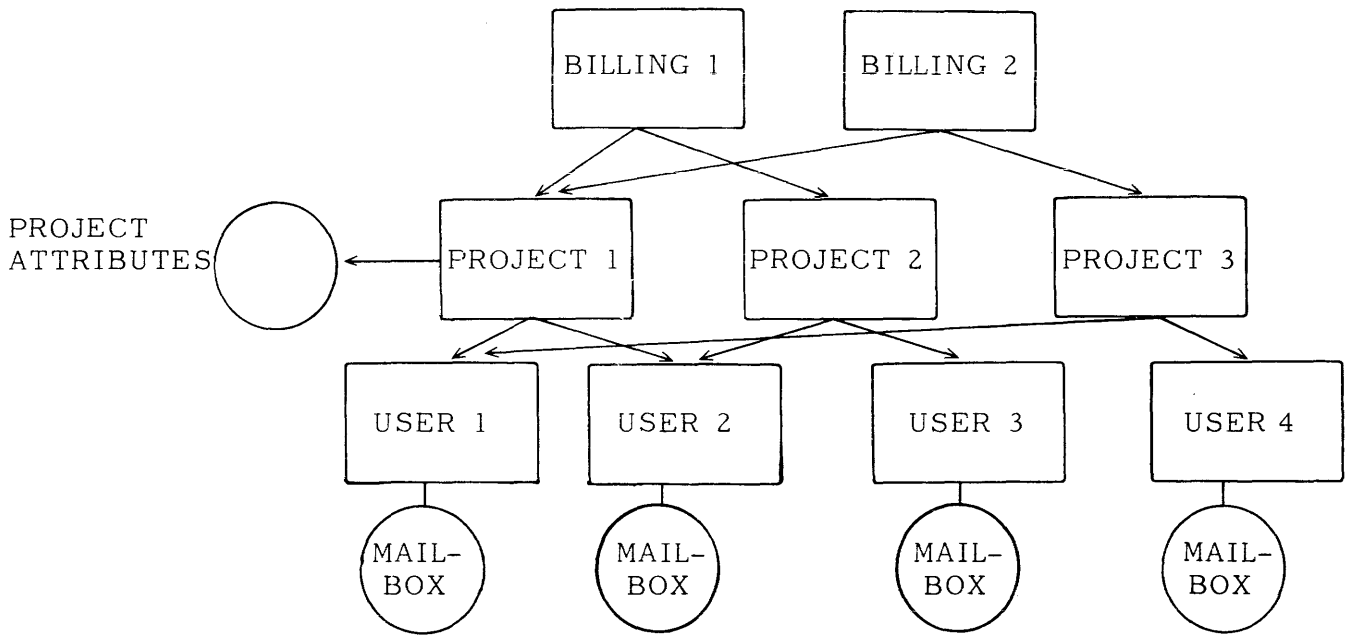


Figure 5-1. Structure of User Registration.

PROTECTION MECHANISMS

Operator Access

The access of an operator to the system is checked by means of a password mechanism. There is one password per registered user and the check is performed by

the listener at log-on time and according to the following rules:

USER-NAME:	is it registered?
PASSWORD:	is it the correct password?
PROJECT: (overrides the default)	is user registered under this project?
BILLING: (overrides the default)	is project registered under this billing mechanism?

Operability Privileges

Operability privileges are IOF specific. They apply to the different operations authorized under IOF. The following operability privileges exist with IOF:

- General User: This user is allowed to enter only the IOF command language as defined, and can only control the jobs and sysout deliveries that he has submitted himself. He cannot act on other jobs or sysout deliveries submitted either by other interactive users or by the master operator on the central station.
- Master Operator: this privilege gives access to the full Operator Control Language (OCL). Also it allows the master operator to control the whole system without any limitations. There is only one Master Operator logged-on at any time, and he is the operator of the Operator station. The physical device identifier may be redefined at system generation time for the Operator Station.

User Registration

The operation that consists of introducing a new user, new project, new billing or modifying operability privileges of a project is the System Administrator's responsibility. Such an administrator is a particular operator who belongs to a project. Since registration involves the GCOS site catalog, all these operations are performed by using the catalog management utilities. Refer to the Catalog Management Manual and the System Management Guide for more details.

SUPPORTED CONFIGURATIONS

The IOF system supports communications procedures for TTY and VIP terminals through the VCAM access method. The network is generated by the Communication Network Configurator utility (CNC) and the BTNS job must be active in order to operate IOF. The VCAM access method makes the physical terminal type transparent to IOF. So all the TTY and VIP type terminals are supported by IOF. The only exception concerns the scrolling mode of CRT type terminals, which is not supported.

In addition to communications support, IOF requires two load modules which must be stored within the SYS HLMLIB library and should be pre-initialized. The two load modules are:

H_IOF load module, which is necessary to operate terminals at the system

command level.
H_LIBMAINT load module, which contains the Library Maintenance and Text Editor conversational processors.
The same load module supports both batch and interactive mode.

The maximum number of users that may log-on under IOF is configurable on-site. The upper limit is the multiprogramming limit which is also configurable on site from 1 to 30 jobs. Default values are: maximum number of logged-on users = 10, the system multiprogramming limit = 15.

APPENDIX A

THE SCANNER PROCESSOR

The scanner processor is able to scan Sysout outputs. These Sysout outputs may be stored either on the public Sysout file, or on a private Sysout file, which can be on disk or tape. The format of Sysout may be either edited or SSF. This facility allows the user to display pages. Pages are identified by their page numbers or by the occurrence of a given character string. This function is useful for checking large Sysout outputs before having the output printed.

Access to outputs available for the user's control is restricted. Only the outputs that have been generated by the jobs the user has submitted can be accessed. The control is performed on the user name. This submitter name is attached to all the jobs submitted by that interactive user and all the outputs generated by those jobs. The submitter name is also kept for cases of jobs submitted by other users through the JCL RUN statement. When a user wants to access a Sysout output, the user-name must be the same as the submitter name attached to the output; otherwise access is denied.

COMMAND LANGUAGE

The Scanner processor is called by using the SCANNER extended JCL statement

```
S:  SCANNER;
C:  ..........
```

The Scanner JCL statement supports the following set of commands:

LIST	- list of output names for a given job.
SCAN	- scan a sysout output which is identified by its output name.
QUIT	- quit from the Scanner processor and return to the system command level.

The SCAN command allows us to scan a Sysout output at page level. Pages are addressed by their page numbers or by a given character string contained in them. The processor also keeps a pointer at the currently addressed page and the user can specify the address of a page relative to the current one.

By convention, when a new page is searched by means of a given character string, the scanning for the required character string is always forward and starts at the first line of the currently addressed page.

A page is addressed in a Sysout by its sequential page number, which may differ from the one printed at the top of the page. A page can also be addressed by its position relative to the currently addressed page. The following set of commands are available for moving backwards and forwards over pages and displaying pages.

#[n]	position at page n
>[n]	n pages forward
<[n]	n pages backward
P[n]	display n pages starting from the current one
=	display the number of the current page

n represents the page number or a number of pages. By convention, the default value of n is 1 and n=$ represents the last page of the Sysout (the first page if moving backward). For instance P$ prints the output from the current page to the last one.

A page can also be referrred to by giving a character string contained in it. The following command is available for this purpose:

/[string]/

The effect of this command is to scan the output from the first line of the current page, until the first occurrence of the string, then print the line containing the string, then point to the first line of the currently addressed page. It is possible to have the printing of the first m occurrences of the string before pointing to the page containing the m-th occurrence, with the command:

/[string]/[m]

Default string means the latest defined one.

Let us consider an example where we want to check the errors generated by a COBOL compilation instead of having the complete compiler listing printed-out. The printout of the compiler output must obviously have been stored.

```
S:      SCANNER;
C:      LIST X42;
        X42:1  JOB_REP  C PR SIZE   803
        X42:2  JOB_OUT  C PR SIZE   80263  MEMBER = 4
C:      SCAN X42:2:3
R:      #5500
R:      /**/ $
        PAGE 005632
        **13-982 IN THE GIVEN VALUE CLAUSE,VALUE MAY BE
        LONGER THAN LENGTH OF DATA ITEM.
        PAGE 006239


        ** 1 5-156 POSSIBLE LEFT TRUNCATION.
        ** 1 5-150 SIGN OF SENDING ITEM NOT MOVED.
        PAGE 006242
        ** 1 5-156 POSSIBLE LEFT TRUNCATION.
        PAGE 006243
        ** 1 5-150 SIGN OF SENDING ITEM NOT MOVED.
R:      Q
C:      QUIT;
S:      CO X42:2
S:.....
.....
.....
```

Let us now consider a second example where we shall scan a private Sysout. The file name is "ACCOUNTING", the media is the 9-track tape "2125".

```
S:      SCANNER;
C:      SCAN ACCOUNTING : 2125 : MT/T9;
R:      #5250
R:      P
        PAGE 5250
                  JOB RECORD
                 ***************
        ACCOUNT = INSTALl
        USER NAME = BTNS
.................................
.................................
R:      Q
C:      QUIT;
S:
```

LIST COMMAND

LIST [ron];

The LIST command lists all the outputs which are in the READY or HOLD states and have been generated by the indicated job. The job must have been submitted by the same user otherwise access is denied.

ron is the Run Occurrence Number of the job whose outputs are to be listed. When ron is omitted, outputs generated by the interactive job are listed.

Information that is displayed, on one or two lines, is formatted as follows:
Xron:index[Output-name][priority] class device SIZE = n1[Xn2] [MEMBER = n3] [[Subfile=] efn]

Xron:index	identifies the output for the SCAN command.
Output-name	name of the output as indicated in the corresponding Sysout statement.
priority	priority of the output in the Output Writer queues (not specified when the default value is used).
class	class of the output.
device	device type (PR,CD.....).
SIZE	gives the size of the output as the number of lines (n1) and the number of requested copies (n2).
MEMBER	number of parts of the JOBOUT output.
[subfile:] efn	in the case of a private Sysout file, the external file name (and the library member name if appropriate).

Here is an example of using the LIST command:

```
C:      LIST X34;
        X34:1 JOB_REP                C PR SIZE=783
        X34;2 JOB_OUT                C PR SIZE=92769  MEMBER = 12
        X34:5                        C PR SIZE 51133
        MYLISTING:MYOUTLIB
        X34:74                     2 A CD SIZE=233
        MYPUNCHFILE
        X34:168                      B PR SIZE 7284
C:
```

SCAN COMMAND

```
SCAN { [member:] efn:media:device-class }
     { ron:index [:jobout index]        } ;
```

The SCAN command provides a set of requests to scan the specified output.

ron is the Run Occurrence Number of the job whose output is to be scanned.

index is a number which identifies an output uniquely from all others generated by the job.

jobout index is a number which identifies an output of the JOBOUT output. This is mandatory when index = 2 (i.e., the JOBOUT output).

The SCAN command only applies to outputs that are known by the Output Writer. An output is always identified by the Run Occurrence Number of a job and an index, plus a jobout index number in the case of a JOBOUT output. The output must be in the READY or HOLD state (i.e., not being created or being printed). Trying to access an output currently being printed gives an unpredictable result. Thus it is recommended to hold an output before scanning it. To access a private Sysout (sequential file or member of a library) this must be specified using the above SCAN command by specifying the device-class, media, efn and member name.

The user must have access rights to the required output; this means a user name which is the same as the submitter name of the job that has generated the output.

We have already seen how a page pointer is used. We should note that improved performances for searches for a given character string can obviously be achieved by skipping any parts of the Sysout where the string cannot occur.

We have also seen how some commands may be used for addressing pages using the current pointer. The general syntax of such a command is as follows:

<request code><parameter>

<request code> is a character from the following set: #,<,>,P,=,/,Q.

<parameter> is optional. It is always a number whose default value is 1. By convention the character $ represents the last page of the Sysout (the first one if moving backward).

The following is a list of the commands that can be used in association with SCAN together with some of the possible return messages:

#[n]	used to position the current pointer at the first line of page n of the Sysout.
>[n]	used to move n pages forward so that the pointer is at the first line of the new page.
<[n]	same as above, but backward.
P [n]	used to display n pages starting from the current one. The return message typically would be: PAGE 008142 ....

PAGE 008143

.....

We should note that a line is automatically displayed on several lines, if it is too long for a single line.

=	used to display the current page number. A typical return message would be: PAGE 008143
/[string]/[m]	used to display the first m lines which contain the given string. The search starts at the first line of the currently addressed page. Default string means the latest defined one. A typical return message would be: PAGE 008143 After that command, the current pointer is set at the first line of the current page.
Q	used to quit the Request level and return to the Processor Command level. The current pointer is lost when we quit.

QUIT COMMAND

QUIT;

The QUIT command, as we have already seen, forces an exit from the Processor Command level and returns operation of the terminal to the System Command level.

HONEYWELL INFORMATION SYSTEMS

Technical Publications Remarks Form

TITLE: SERIES 60 (LEVEL 64) GCOS INTERACTIVE OPERATION FACILITY

ORDER NO. AQ60, REV 0

DATED SEPTEMBER 1978

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. ☐

FROM: NAME _____ **DATE** _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

PLEASE FOLD AND TAPE –
NOTE: U. S. Postal Service will not deliver stapled forms

FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

Honeywell