Global 16-bit Development System
Job Management Manual
Version 8.1

TABLE OF CONTENTS

APPENDICES

1. Introduction and Overview

Job management allows you to run Global System Manager commands and application programs under the control of a job dialogue table, held in memory, which supplies predetermined responses to prompts, thus minimising operator interaction.

In standard job management you create a job file which, when executed, establishes a job dialogue table in memory and then initiates it. Additional flexibility can be achieved by defining parameters whose values you supply when the job file is run.

There is also an advanced form of job management in which you produce a tailored job initiator to replace the standard job file. Your initiator will build a job dialogue table to your precise requirements and will initiate it when complete.

You will find uses for job management in all operationally repetitive situations. For example:

- Program development, where compilations and linkage edits can be automatically executed;
- Installing software products, e.g. building operational volumes from distribution volumes;
- Running Global System Manager command programs as part of an application system.

1.1 Terminology

Throughout this manual the term **job file** refers to a job management control file as built by $JOB, and the term **job** (as in end-of-job routine) refers to the activity which takes place between the time that a program is executed from a ready prompt until the following ready prompt.

1.2 The Job Description

To use standard job management, you must first of all prepare a job description. This is similar in many ways to a Global Cobol source file and, indeed, resembles a rather strange COBOL program since its parameter division and dialogue division are comparable in many ways to a data division and procedure division.

In the parameter division you describe, and define prompts for, any input that the **operator** must provide each time that the job file is run in order to complete the information supplied in the dialogue division. For example, in a job description controlling program compilation you would probably decide that the name of the Global Cobol source file should be a run-time parameter described in the parameter division. Normally the operator will supply values for all parameters in sequence and then the execution of the job file will start. However, any parameter can be defined to be conditional: The operator can then key a special response to its prompt which causes the prompting for the remaining parameters to be by-passed and job file execution to begin.

Global System Manager prompt	Reference

LIBRARY library-id REQUIRED ON unit:	$MONITOR
PROGRAM program-id REQUIRED ON unit:	$MONITOR
PROGRAM program-id IN LIBRARY library-id REQUIRED ON unit:	$MONITOR
PLEASE KEY PASSWORD FOR file-id:	$MONITOR
PLEASE MOUNT volume-id ON unit AND KEY <CR>:	$MONITOR
PLEASE ASSIGN unit-id:	$MONITOR
* error ON description - unit - RETRY?:	$MONITOR
MOUNT BACKUP backup-id ON description-unit AND KEY <CR>:	$SAVE
CONTINUE WITH NEW BACKUP VOLUME?:	$SAVE
$34 CONTINUE?:	$TDUMP
$34 DESTROY tape-id?:	$TDUMP
$34 MOUNT TAPE tape-id:	$TDUMP
$34 TOO MANY ERRORS - RETRY?:	$TDUMP
$34 MISSING DATA (nnnn BLOCKS) - RETRY?:	$TDUMP
$34 FILE file-id ON unit n MISSING BLOCKS	$TDUMP
$43 SPECIFY NEW COMPILATION UNIT:	$COBOL
$43 SPECIFY NEW LISTING UNIT:	$COBOL
$44 SYSTEM LIBRARY library REQUIRED ON unit:	$LINK
Activate debugger, key D for diagnostics prompt, H for help, <ESC> to exit:	$DEBUG
$54 SPECIFY NEW LISTING UNIT:	$XREF
$78 DESTROY file-id ON UNIT unit?:	$V
$78 DESTROY unit?:	$V
$78 DESTROY volume-id?:	$V
UNIT unit forms-control-message:	$MONITOR
$91 BREAK?:	$MONITOR
comma prompt on base line	$MONITOR

* This prompt only bypasses job management if it follows a program check.

Table 1.2 - Global System Manager Prompts which bypass Job Management

In the dialogue division header you may specify that the job file is to be executed without any dialogue appearing at the operator console, once the parameters have been accepted. In the dialogue division body you anticipate all the prompts that the subject program (or programs) will cause to be output, and supply responses for them. A response can either be a predetermined character string, a parameter, or a control sequence such as <CTRL A>, <CTRL B>, <CTRL C>, <ESCAPE> or <CR>.

Because certain Global System Manager prompts, shown in Table 1.2, would be very hard to predict when creating a job description, they are specially identified and, if they occur, job management is bypassed and the operator is prompted in the normal way, without using the dialogue you have created. If you reply <ESCAPE> to one of these prompts and the program does not suppress the use of the escape key, job management will be immediately terminated with the message:

```
$17 JOB MANAGEMENT TERMINATED
```

followed by a ready prompt. In addition, Global System Manager commands and system routines have been modified so that when job management is in control the prompts:

```
FILE ALREADY EXISTS - DELETE?:
```

and:

```
NEXT PAGE?:
```

are automatically provided with the response Y, whenever they occur. Because of these simplifications, it is easy to supply the necessary responses to any set of programs. You start by indicating the response to the ready prompt which will cause the first program controlled by the job description to be entered. As well as responses, you may specify pause prompts and messages in the dialogue division. When a pause prompt is met the text it contains is displayed and Global System Manager waits until the operator hits any key. Pause prompts are usually provided to allow diskettes to be changed at the correct time. Messages are used to provide information at the console when no operator reply is required.

Conditional parameters may be tested within the dialogue division, and job file execution will be terminated immediately if the special operator input which terminates parameter prompts was supplied for that parameter (or a previous one) when the job file was initiated.

In summary, job descriptions are equivalent in many ways to the catalogued procedures used on main-frame computers. Section 2.1 explains in detail how to create job descriptions.

Error Prompt or Message	Reference	Note
INVALID - REINPUT	$MONITOR	
MEMBER NOT FOUND	$LIB	
NOT FOUND	$MONITOR	
NOT FOUND OR WRONG TYPE	$MONITOR	
RECORD SIZE TOO LONG	$L	
INPUT REQUIRED	$MONITOR	1
INVALID	$MONITOR	1
TOO LARGE	$MONITOR	1
TOO LONG	$MONITOR	1
$43 NUMBER OF ERRORS non-zero	$COBOL	2
$43 NUMBER OF WARNINGS non-zero	$COBOL	2
$44 LINK ABORTED - reason etc.	$LINK	
$52 NUMBER OF ERRORS non-zero	$JOB	
$54 NUMBER OF ERRORS non-zero	$XREF	2
$54 NUMBER OF WARNINGS non-zero	$XREF	2
$64 TOO MANY ENTRIES	$CATAL	
$69 ASSIGNMENT TABLES FULL	$A	
$90 program-id TOO LARGE	$MONITOR	

Note 1 These messages will not cause termination if they appear because the operator has supplied erroneous input to a prompt which bypasses job management.

Note 2 Termination can be prevented in these cases by use of an appropriate compilation or cross-reference option.

Table 1.5 - Typical Global System Manager Prompts and Messages Causing Job Management Termination

1.3 The Job File Builder ($JOB)

Once you have created a job description, you run the $JOB command to produce a job file from it. The job file is actually a program file,

so it can be saved, either as an individual file or as a member of a program library, and invoked by keying its name in response to the ready prompt or by chaining to it from an application program. A job file may also be run under job management.

The data division of the job file contains a parameter table and dialogue table, which are compressed versions of the job description's parameter division and dialogue division. Each job file occupies 4K or more bytes of disk space, depending on the size of its parameter prompt area and the amount of dialogue it contains.

The $JOB command also produces a print file known as the job listing. This is a validated version of the job description. It contains information such as the date on which the job file was created and the number of bytes of the dialogue table created. If the job description is faulty, error messages will appear on the job listing and no job file will be produced.

Section 2.2 describes in detail how to use the $JOB command to create a job file and Appendix A contains sample job listings.

1.4 Running a Job File

The usual method of running a job file is to key its name in response to the ready prompt. It is also possible for one job file to call another or for an application program to invoke a job file. In all cases the job file program is loaded, and control is passed to its entry point, the job initiation routine.

The job initiation routine firstly examines its parameter table and, if a parameter division was present in the job description, prompts the operator for the required input, which is then used to complete the dialogue table. If the job file was invoked from another job file, the required parameter values may be supplied directly by the calling job file.

Next the dialogue table, incorporating the parameter values supplied for this execution of the job file, and the job manager (see below) are moved to the user stack at the top of the Global Cobol memory region. The user stack will have to be extended to accommodate them, thus reducing slightly the user area available to the subject programs.

The job initiation routine now logically replaces the console input routine by the job manager. This means that whenever a program requires input, the job manager, rather than the normal console input routine, will receive control. The job manager will then supply input from the dialogue table stored in the user stack, rather than requiring operator intervention.

Finally the job initiation routine returns to the monitor, which issues a ready prompt in the normal way. Since the job manager is in charge of all console input, it supplies the response to this - the name of the first subject program to run under job management - from the dialogue table in the user stack.

1.5 Global System Manager under Job Management

Once job management is active, Global System Manager continues to extract responses from the dialogue table, one by one, until either

the end of the table is met, or one of the messages or prompts summarised in Table 1.5 appears. If you examine these carefully you will see that they indicate that one of the following types of error has occurred:

- An incorrect response has been obtained from the dialogue table. This response may have been supplied as a parameter by the operator. Typically this might result in an INVALID prompt or FILE NOT FOUND OR WRONG TYPE message.

- The subject program has met with an irrecoverable error, for example output file space is exhausted.

Normally, when the end of the dialogue table is reached without there being an error, the job manager automatically de-activates itself by restoring the true console input routine so that the operator can continue to communicate with Global System Manager in the normal way. The area of user stack used to hold the job manager and dialogue table is released.

If an error message or prompt from Table 1.5 appears, job management is immediately terminated with the message:

```
$17 JOB MANAGEMENT TERMINATED
```

and control returns to the monitor.

If the dialogue table supplies the response $E to the ready prompt, then the $E command is run and end-of-session processing takes place. This processing includes de-activation of job management: Global System Manager waits for operator input to start the next session.

The Global System Manager file utility, $F, and library utility, $LIB, both have special facilities which enable them to perform volume-id checking when executing under job management. $F provides the I and O instructions which check that the required volume is mounted on the input or output device respectively before proceeding with the next file operation. There is also an IO instruction which combines the functions of I and O. $LIB provides the I instruction which checks that the required volume is mounted on the input device, and if the special response <CTRL C> is supplied to the $95 TARGET LIBRARY: prompt then $LIB will check that the required volume is mounted on the output device. These facilities are fully described in the Global Utilities Manual ($F), and Global Development Toolkit Manual ($LIB). Examples of the use of each is given in the job listing L.INSPR in Appendix A of this manual.

The prompts and messages of Tables 1.2 and 1.5 may vary with Global System Manager versions and should therefore be considered as representative. More generally, Global System Manager command program dialogue may also vary with Global System Manager versions. These differences will be kept to the minimum consistent with a continuously improving software product, but **if you are running Global System Manager command programs under job management you must be prepared for the possibility that your job files may need to be upgraded to move to a later Global System Manager version.**

1.6 Advanced Job Management

Chapter 3 describes the more advanced facilities available under job management, and a brief summary is given here.

While existing applications may be run unchanged under job management, new applications can take advantage of special controls which are provided to improve operator interaction and the handling of error situations.

It is possible to call one job file from another, passing parameters if required, and to initiate job dialogue from an application program.

Using the JOB$ system routine you can create your own tailored job initiator which can assemble a dialogue table from short sequences of dialogue and then initiate it.

1.7 Restrictions

The size of a job dialogue table is limited only by the size of the user area and the size of the programs to be run under its control. The dialogue table is loaded into the user stack at the top of memory, thus reducing the amount of memory available to its subject programs. The job listing reports the size of user area needed to initiate each job file, and the size of user stack needed during its execution.

A maximum of 20 parameters may be supplied for the job description.

Single character console input operations, as performed by CHAR$, and console status testing operations, handled by CHECK$ and TEST$, always bypass job management. Dialogue therefore cannot be supplied to these routines from the job dialogue table: they always access the console directly. For further information refer to the Global Development Screen Presentation Manual.

2. Creating a Job File

To create a job file you must first produce a text file known as the job description, and then use the $JOB command to build the job file from this job description. This chapter describes the process in detail.

2.1 The Job Description

A job description is a text file containing lines of up to 72 characters. It can be created and maintained by the $EDIT command. The skeleton form of the job description is as follows:

```
JOB job-id title

[PARAMETER DIVISION

parameter definition lines]

DIALOGUE DIVISION [(SUPPRESS)]

dialogue definition lines,
optional pause prompt lines,
optional message lines
and optional conditional exit lines,
in any order

ENDJOB
```

Each line is in free format, with leading blanks and tabs before the first significant character being ignored. Within the body of a line, a blank, tab, or any combination of blanks and tabs may serve as a separator between different job description elements. Optional constructs are represented enclosed in square brackets.

Figure 2.1 shows a typical job description which contains examples of all the main facilities, and you will find it helpful to refer to it to supplement the explanation which follows. It performs a fairly straightforward program preparation sequence, with conventional unit assignments.

```
JOB CLXMP COMPILE, LINK & XREF MAIN PROGRAM
*
* THIS JOB DESCRIPTION COMPILES THE SOURCE OF A MAIN PROGRAM,
* LINKAGE EDITS IT WITH THE SYSTEM LIBRARIES,
* AND OPTIONALLY PRODUCES A CROSS-REFERENCE LISTING.
* ALL LISTINGS ARE PRODUCED ON THE LOGICAL PRINTER, $PR.
*
PARAMETER DIVISION
*
 &1    X(6) PROGRAM NAME (SUFFIX ONLY)
 &2    X(3) USER UNIT
 &3    X(1) ? XREF REQUIRED? (Y/N)
*
DIALOGUE DIVISION
*
* FIRSTLY, COMPILE THE PROGRAM
+COMPILATION STARTING+
GSM READY:<CR>
GSM READY:$COBOL
$43 SOURCE:&1 UNIT:&2
$43 COMPILATION UNIT:<CR> SIZE:<CR>
$43 COPY LIBRARY:<CR>
$43 LISTING UNIT:<CR>
```

```
$43 COMPILER OPTION:<CR>
* THEN LINK THE PROGRAM WITH THE SYSTEM LIBRARIES.
+LINKAGE EDITING STARTING+
GSM READY:<CR>
GSM READY:$LINK
$44 LINK:&1 UNIT:&2
$44 LINK:<CR>
$44 PROGRAM:<CR> UNIT:<CR>
$44 LISTING UNIT:<CR>
$44 LINK OPTION:<CR>
*CHECK IF XREF HAS BEEN SELECTED
?3
* PAUSE TO ASK IF XREF IS REQUIRED.
-CONFIRM XREF REQUIRED, <ESC> OTHERWISE-
* NOW PRODUCE THE CROSS-REFERENCE.
GSM READY:<CR>
GSM READY:$XREF
$54 SOURCE:&1 UNIT:&2
$54 COPY LIBRARY:<CR>
$54 LISTING UNIT:<CR>
$54 X-REFERENCE OPTION:<CR>
*
ENDJOB
```

Figure 2.1 - Example Job Description

2.1.1 The Job Line

The job line, beginning with the word JOB, must be the very first line of the job description. Its format is:

```
JOB job-id title
```

The job-id is really only present to enable you to document the name that you intend to give to the job file when it is produced by the $JOB command from this job description. The name of the job file is actually supplied from information keyed to the $JOB command, but you will normally choose to make this the same as the job-id. A job-id is usually six characters or less, and the job description source file should be named S.job-id (e.g. S.CLXMP) in the case of the example of Figure 2.1. The job-id is terminated by the first blank or tab following it in the job line.

The title begins with the first character following the job-id which is not a blank or tab. It can be up to 30 characters in length and can contain embedded blanks. The title will appear on the screen when the job file is run, and is stored in the program header record of the job file, so that it can become a member title if the job file is later incorporated into a program library. If the title is completely blank it will not be displayed when the job is run.

2.1.2 Comment Lines

A comment line, whose first non-blank character is an asterisk, can appear anywhere within the job description, but may not precede the job line. The comment is output unchanged to the job file listing, but is otherwise ignored.

Note that comments in a job description should only appear on lines by themselves. It is **not** recommended to comment other job description lines by coding to the right an asterisk followed by explanatory text, as in Global Cobol. This is because an asterisk may well be a significant character in a dialogue definition or pause prompt line.

2.1.3 The Parameter Division

If the entire dialogue for the program (or programs) to be run by a job file can be specified when the job description is created, no parameter division is needed. However, if the operator is to be able to vary selected parts of the dialogue when the job file is run, then a parameter division must be coded. The parameter division of the example of Figure 2.1 gives the operator control of the program name, user work unit and cross-reference selection, all other aspects of the program preparation process being invariant for all executions of the job file.

When present the division begins with the header:

```
PARAMETER DIVISION
```

which must be coded on a new line. It is terminated by the header:

```
DIALOGUE DIVISION [(SUPPRESS)]
```

also on a new line.

2.1.4 Parameter Definition Lines

When a parameter division is supplied, one or more parameter definition lines must be coded between the delimiting headers. Each line is of the form:

```
parameter-name format [? ] prompt-message
```

The parameter-name consists of an ampersand immediately followed by one or two digits representing a parameter number in the range 1 to 20 inclusive (e.g. &1). It is an error to use the same parameter-name on more than one parameter definition line.

The format resembles the picture clause of a character or display numeric item, except that the word PIC is not used. The format information is used to check the operator input supplied for the parameter when the job file is initiated. Typical valid formats are:

```
S9(3)
X(2)
9(4,3)
X
9
```

A format cannot contain a blank or tab, so for example:

```
9(4) COMP
```

is **not** a valid format. There is in any case no requirement to support computational parameters since all console input must be either character or display numeric.

The optional ? indicates that the parameter is conditional. If <CR> is keyed to a conditional parameter then it, and all subsequent parameters, will be set to <CR>. No further prompts will be displayed and the job file will be initiated immediately. Later, the running job file will inspect the value of the conditional parameter and will terminate prematurely if the parameter is <CR>. The format X(1) has a special meaning when it is used with a conditional parameter: an operator reply of N has exactly the same effect as the <CR> input

described above. This facility enables more meaningful replies to be made when entering conditional parameters, and an example of this form of conditional parameter appears in Figure 2.1. Note that the format X used with a conditional parameter does not have the special meaning associated with the format X(1) : N is treated as a normal reply, not as a synonym for <CR>.

If you reply <CTRL A>, <CTRL B> or <CTRL C> to any prompt this reply will be accepted and stored as the parameter value. Similarly you may terminate a reply with one of these special responses, for example 123<CTRL A>. If you reply <CR>, and the ? has not been specified, then the parameter value will simply be set to <CR>.

The prompt-message begins with the first non-blank character following either the format or the ? if present, and is terminated by the end of the line. Prompt-messages are stored by the $JOB command in the parameter table in parameter number order. When the job file is run, they are output, followed by a colon character, as prompts requesting input of the type governed by the associated format. For example, suppose a job description contains the following parameter division:

```
PARAMETER DIVISION
&1    9(2,2)      RATE
&2    X(8)  ACCOUNT CODE
DIALOGUE DIVISION
```

Then, when the job file is run, the following dialogue might take place:

```
RATE:122 INVALID - REINPUT
:12.2
ACCOUNT CODE:AB40798
```

The INVALID - REINPUT prompt appears because the operator input did not satisfy the format information associated with the first parameter. Note that during job initiation the validation of operator input is limited to that defined by the associated format information, and the job file may still fail if the accepted parameter value is later rejected by more thorough validation by the program to which the value is supplied as input. Thus, for example, the account code supplied above, while conforming to the format X(8), may not be acceptable to the program to which it is to be supplied.

2.1.5 The Dialogue Division

A dialogue division must always be present in a job description. It begins with the header:

```
DIALOGUE DIVISION [(SUPPRESS)]
```

and ends with the trailer:

```
ENDJOB
```

Both must be coded on a new line. The alternative spelling DIALOG is acceptable in the header.

If the (SUPPRESS) option is coded, the dialogue of the job file will not in general be displayed on the console when the job file is run. Once parameters have been accepted in the normal way, all dialogue is suppressed except for the following:

- pause prompts (see 2.1.7);
- messages (see 2.1.8);
- program prompts which bypass job management (see Table 1.2 for Global System Manager commands and section 3.1 for application programs).

Suppression of dialogue begins as soon as the job file has been loaded and its parameters (if any) have been supplied, and continues until the last response of the dialogue table has been supplied by the job manager to a subject program.

Between the header and trailer may appear any number of dialogue definition lines, pause prompt lines, message lines, and conditional exit lines. Syntactically, these may be in any order, but of course the sequence of lines is very important, since their arrangement determines the order in which responses are supplied to prompts, and the times at which pause prompts appear and conditional parameters are checked.

2.1.6 Dialogue Definition Lines

There must always be at least one dialogue definition line present, since the first one encountered specifies the response to be made to the ready prompt output by the monitor immediately following job management initiation. The format of each dialogue definition line is:

```
text[:response text...]
```

Each text consists of zero or more characters, including blanks and tabs, but excluding the colon character which is used to introduce the response. The first text on the line must not begin with an asterisk, question mark, plus sign or minus sign, nor must it start with the word ENDJOB. In the interests of good documentation, each text item should normally correspond exactly to the display that the subject program will produce at this stage of the dialogue. However, **no information about the text is placed in the dialogue table by the $JOB command**. Note that text may appear on a line by itself.

Each response is coded immediately to the right of a colon character. It may be either an input string (i.e. a number of characters to be supplied in place of operator input), or one of the special responses listed in Table 2.1.6. An input string may be directly followed by one of the special terminators <CTRL A>, <CTRL B>, <CTRL C> or <ESCAPE>. Thus the response:

```
:12<CTRL C>
```

supplies the value 12 and the terminator <CTRL C> to the subject program.

The following rules apply when specifying responses:

- Each response must be terminated by a blank, tab, or end of line;
- The special responses must be coded exactly as shown in the first column of Table 2.1.6. If a parameter name is used it must have been introduced in a previous parameter definition line within the parameter division. For a parameter name response, the value passed

to the subject program by the job manager will normally be the value supplied to the corresponding parameter prompt-message. However, if the parameter is conditional and a <CR> value was supplied for it, or if a <CR> value was supplied for an earlier conditional parameter, then the special response <CR> will be passed to the subject program;

- An input string must not begin with ampersand or colon, nor may it contain embedded blanks or tabs;

- If you need to specify an input string containing blanks, then you must code an @ character whenever a blank is required. All @ characters will be converted to blanks in the strings stored in the dialogue table by the $JOB command.

Special response	Meaning
&n	Parameter n, where 1 ≤ n ≤ 9
&nn	Parameter nn, where 10 ≤ nn ≤ 20
<CTRL A>	Null string, consisting simply of CTRL A
<CTRL B>	Null string, consisting simply of CTRL B
<CTRL C>	Null string, consisting simply of CTRL C
<ESCAPE> or <ESC>	Null string, consisting simply of ESCAPE
<CR> or <NULL>	Null string, consisting simply of RETURN

Table 2.1.6 - Special Responses

The last rule means that you cannot actually provide an input string containing an @ character as a response, since any such character will automatically be converted to a blank.

Whenever a program completes execution under Job Management, control is returned to the monitor and the GSM READY prompt will appear so that another program may be executed.

2.1.7 Pause Prompt Lines

The format of a pause prompt line is:

```
-prompt-
```

where prompt consists of one or more ASCII characters. The prompt may contain spaces or tabs, but cannot include a minus sign, since this character is used as the start and end delimiter.

If, when the job manager attempts to retrieve the next response from its dialogue table, it encounters a pause prompt, the prompt is output to the screen on a new line without the leading minus sign and with the trailing minus sign replaced by a colon. The job file now waits until the operator keys <CR> or <ESCAPE>. If the operator keys <ESCAPE>, for example because he or she cannot find a required diskette, then job management will be terminated. If the operator keys <CR> a new line will appear at the screen and processing will

continue. Thus the following dialogue division statements, taken from the example of Figure 2.1:

```
...
-CONFIRM XREF REQUIRED, <ESC> OTHERWISE-
GSM READY:<CR>
GSM READY:$XREF
$54 SOURCE: ...
```

would cause the following dialogue to take place when their job file was run:

```
...
GSM READY:
CONFIRM XREF REQUIRED, <ESC> OTHERWISE:<CR>
GSM READY:$XREF
$54 SOURCE: ...
```

This pause prompt gives the operator the opportunity to change diskettes if necessary, and as soon as he or she replies <CR> the job manager continues by loading $XREF.

Note the use in the example of the dialogue definition line:

```
GSM READY:<CR>
```

Its purpose is merely to improve the appearance of the dialogue displayed on the screen when the job file is run. If this line is omitted, the screen dialogue will appear as:

```
GSM READY:
CONFIRM XREF REQUIRED, <ESC> OTHERWISE:<CR>
$XREF
$54 SOURCE...
```

because the job manager displays the pause prompt immediately before displaying the next response. This technique applies also to the message facility described in 2.1.8.

A pause prompt will always appear on the screen when a job file is run, even if the job file has suppressed dialogue.

2.1.8 Message Lines

The format of a message line is:

```
+message+
```

where message consists of one or more ASCII characters. The text may contain spaces or tabs, but cannot include a plus sign, since this character is used as the start and end delimiter.

If, when the job manager attempts to retrieve the next response from its dialogue table, it encounters a message, the message is output to the screen on a new line without either the leading or trailing plus sign, and followed by a new line. The job file then continues processing.

Thus the following dialogue division statements, taken from the example of Figure 2.1:

```
...
+COMPILATION STARTING+
```

```
GSM READY:<CR>
GSM READY:$COBOL
$43 SOURCE: ...
```

would cause the following dialogue to be displayed when their job file was run:

```
...
GSM READY:
COMPILATION STARTING

GSM READY:$COBOL
$43 SOURCE: ...
```

A message will always appear on the screen when a job file is run, even if the job has suppressed dialogue.

2.1.9 Conditional Exit Lines

The format of a conditional exit line is:

```
?n
```

or:

```
?nn
```

where n or nn is the number of a parameter defined as conditional in the parameter division. If, when the job manager attempts to retrieve the next response from its dialogue table, it encounters a conditional exit, and either the value keyed for the corresponding parameter was <CR> or a <CR> value was supplied for an earlier conditional parameter, it returns to the monitor without processing any remaining dialogue in its table. If the conditional parameter was not <CR>, the job manager proceeds immediately to the following entry in its dialogue table.

2.1.10 Executing the Example Job File

Suppose the $JOB command (described in section 2.2) has been run to create the CLXMP job file from the example job description of Figure 2.1. Then the dialogue shown in Figure 2.1.10 might appear were that job file to be run. (The numbers on the right hand side are not part of the dialogue, but are simply used as reference line numbers in the description which follows.)

You begin running the job file by supplying its file-id (job-id) in response to the ready prompt (line 0), and the job title is then displayed (line 2). You are then prompted for the three parameters defined in the parameter division (lines 4,5 and 6). Once this information has been keyed, job management is initiated and Global System Manager obtains all responses from its internal dialogue table until the assign prompt in line 15 appears. This is one of the prompts that bypass job management. It is output at this stage because the <CR> response to the $43 LISTING UNIT: prompt means that the default listing unit $PR, the logical printer, is to be used by the compiler. In this example $PR is unassigned at this stage, and so the prompt appears.

Once you have keyed your reply, the job file will run automatically without your intervention until the pause prompt in line 35. When you have replied to this, the job file will continue to run automatically until the end of the dialogue is reached. The final ready prompt (line 47) is not under job management.

During the compilation and linkage edit, existing compilation and program files (C.TEST and TEST respectively) were replaced by new versions. The fact that replacement took place is indicated by:

```
FILE ALREADY EXISTS - DELETE?:Y
```

appearing in lines 12 and 29.

You will notice how the <CR> response appears at the screen as a single blank. <CTRL A>, <CTRL B> and <CTRL C> (not used in the example) appear as an up-arrow followed by an A, B or C respectively, but nothing appears for <ESCAPE>.

```
GSM READY:CLXMP                                                  0
                                                                 1
COMPILE, LINK & XREF MAIN PROG                                   2
                                                                 3
PROGRAM NAME (SUFFIX ONLY):TEST                                  4
USER UNIT:205                                                    5
XREF REQUIRED? (Y/N):Y                                           6
GSM READY:                                                       7
COMPILATION STARTING                                             8
                                                                 9
GSM READY:$COBOL                                                 10
$43 SOURCE:TEST UNIT:205                                         11
$43 COMPILATION UNIT: SIZE: FILE ALREADY EXISTS - DELETE?:Y      12
$43 COPY LIBRARY:                                                13
$43 LISTING UNIT:                                                14
PLEASE ASSIGN $PR:500                                            15
$43 COMPILER OPTION:                                             16
$43 COMPILING                                                    17
$43 END OF FIRST PASS                                            18
$43 NUMBER OF ERRORS     0                                       19
$43 NUMBER OF WARNINGS   0                                       20
$43 COMPILATION COMPLETED                                        21
                                                                 22
GSM READY:                                                       23
LINKAGE EDIT STARTING                                            24
                                                                 25
GSM READY:$LINK                                                  26
$44 LINK:TEST UNIT:205                                           27
$44 LINK:                                                        28
$44 PROGRAM: UNIT: FILE ALREADY EXISTS - DELETE?:Y               29
$44 LISTING UNIT:                                                30
$44 LINK OPTION:                                                 31
$44 LINKAGE EDITING                                              32
$44 LINKAGE EDIT COMPLETED                                       33
GSM READY:                                                       34
CONFIRM XREF REQUIRED, <ESC> OTHERWISE:<CR>                      35
                                                                 36
GSM READY:$XREF                                                  37
$54 SOURCE:TEST UNIT:205                                         38
$54 COPY LIBRARY:                                                39
$54 LISTING UNIT:                                                40
$54 X-REFERENCE OPTION:                                          41
$54 SOURCE PASS                                                  42
$54 NUMBER OF ERRORS     0                                       43
$54 NUMBER OF WARNINGS   0                                       44
$54 X-REFERENCE COMPLETED                                        45
                                                                 46
GSM READY:                                                       47
```

Figure 2.1.10 - Screen Dialogue when Job File CLXMP is run

2.2 Building a Job File using $JOB

The $JOB command is used to create a job file from a job description, and to produce a job listing. The job file takes the form of an

executable program which can be run by supplying its name in response to the ready prompt in the normal way. The job listing is simply an annotated printout of the job description, which may contain warning or error messages if the job description is suspect or faulty. If errors are found no job file is produced.

2.2.1 The Job Description Prompt

$JOB begins by prompting you for the file-id and unit-id of the job description to be processed. If you do not explicitly key a prefix when you input the file-id, $JOB will assume that the file is conventionally named and will append the S. prefix by default. For example:

```
GSM READY:$JOB
$52 JOB DESCRIPTION:CLXMP UNIT:205
$52 JOB FILE:
```

indicates that the operator wishes to process job description S.CLXMP on unit 205.

2.2.2 To Quit

To quit and return control to the monitor, simply key <ESCAPE> to any prompt output by $JOB.

2.2.3 The Job File Prompt

When the job description has been specified you will be prompted for the file-id and unit-id of the job file to be produced. For example:

```
$52 JOB FILE:TCLXMP UNIT:204
$52 LISTING UNIT:
```

If <CR> is keyed in response to JOB FILE: the suffix of the job description is taken as the file-id of the job file.

If <CR> is keyed in response to UNIT: the unit used is the one specified for the job description.

2.2.4 The Listing Unit Prompt

Once job file details have been specified the listing unit prompt is displayed:

```
$52 LISTING UNIT:205
$52 JOB DESCRIPTION NOW BEING PROCESSED
```

If the suffix of the job description is CLXMP then the above example will cause the listing to be written to file L.CLXMP on unit 101. If the listing unit is a direct access device, the file will initially be allocated the maximum contiguous space available. Any unused space will be returned once $JOB completes.

If <CR> is keyed in response to the listing unit prompt, the file will be placed on unit $PR, the logical printer.

If <CTRL A> is keyed in response to the listing unit prompt, no listing will be produced.

2.2.5 The Option Prompt

You are next prompted:

```
$52 OPTION:
```

to which the replies TW, NTW and <CR> may be given. TW causes the job to be created only if no warnings are found. NTW or <CR> allow a job to be created even if there are warnings.

2.2.6 Processing

Once you have satisfied the option prompt, the message:

```
$52 JOB DESCRIPTION NOW BEING PROCESSED
```

appears and $JOB begins to validate the job description file and output the job file and listing. Once this is finished the following message may appear if $JOB has detected one or more suspect, though not fatal, conditions:

```
$52 NUMBER OF WARNINGS nnnn
```

where nnnn is the number of warning messages output to the job listing. Finally, one of the following two messages appears:

```
$52 NO ERRORS FOUND - JOB FILE CREATED
```

or:

```
$52 NUMBER OF ERRORS nnnn - JOB FILE NOT CREATED
```

and control returns to the monitor, which displays its ready prompt. The first of these indicates that, although warnings may have been flagged, there have been no serious error conditions and a job file has been produced. The second message shows that serious errors have occurred and in consequence no job file has been created.

2.2.7 The Job Listing

A complete listing of the source of the job description is produced, showing the number of bytes of dialogue table generated for each line of dialogue. At the end of the listing appear the job file statistics, showing the size of user area needed to initiate the job file, and the amount of user stack which is needed during its execution. This second figure is the amount by which the standard user area will be reduced during the running of the job file.

2.2.8 Operating Notes

If the listing file is being written to direct access storage and it becomes full, $JOB displays the error message:

```
$52 JOB LISTING FILE SPACE EXHAUSTED
```

and returns to the monitor, which displays its ready prompt. A partially completed file may then be in existence and can be deleted using the $F DEL instruction.

A partially completed file may also result, and require deletion, if $JOB is terminated by an irrecoverable I/O error.

2.2.9 Example 1 - Creating Standard Job File CLXMP

The following dialogue takes place when you process job description S.CLXMP to create a conventionally named job file, CLXMP. The job listing is written to your logical printer, $PR. Both the job description and the job file occupy the volume on unit 204:

```
GSM READY:$JOB
```

```
$52 JOB DESCRIPTION:CLXMP UNIT:204
$52 JOB FILE:<CR> UNIT:<CR>
$52 LISTING UNIT:<CR>
$52 JOB DESCRIPTION NOW BEING PROCESSED
$52 NO ERRORS FOUND - JOB FILE CREATED
GSM READY:
```

2.2.10 Example 2 - Testing a Modification to Job File CLXMP

In this example, having modified S.CLXMP slightly, we produce a job file named TCLXMP on 205 and write the job listing, L.CLXMP, to 204:

```
GSM READY:$JOB
$52 JOB DESCRIPTION:CLXMP UNIT:204
$52 JOB FILE:TCLXMP UNIT:205
$52 LISTING UNIT:204
$52 JOB DESCRIPTION NOW BEING PROCESSED
$52 NUMBER OF ERRORS 2 - JOB FILE NOT CREATED
GSM READY:
```

No job file has been output, since the job description was faulty.

3. Advanced Job Management Facilities

3.1 Applications under Job Management

Most existing applications can be run under job management with no change. However, in some cases you may need to indicate, as Global System Manager does, that certain messages or prompts indicate error conditions.

In rare circumstances you may also require that the response to a prompt bypasses job management and comes directly from the operator. The Global Cobol BELL statement allows you to terminate job management in error situations in a tidy manner, and the job manager control flag, the system variable $$JCL, is provided to allow you to bypass job management when required.

3.1.1 Terminating Job Management using the BELL Statement

The BELL statement, described in the Global Development Screen Support Manual, is usually coded when an error has been detected in console input. Its effect is to sound the console bell (if there is one) and to clear the console input buffer of any type-ahead data, so that correct input can be supplied by the operator. When the BELL statement is executed by a program which is running under job management, it has the additional affect of forcing all subsequent console output to be displayed (even if the job has suppressed dialogue) until the next time that console input is requested, when the program will be terminated with the message:

```
     $17 JOB MANAGEMENT TERMINATED
```

By structuring your parameter validation logic as follows:

```
     DISPLAY message
REINPUT.
     ACCEPT reply
     IF invalid
          BELL
          Display error-message
          GO TO REINPUT
     ELSE
          program continues with good input
     END
```

you can create programs which function consistently whether job management is in control or not. When job management is not in control, the bell will be sounded and the operator re-prompted with an error message if an invalid reply is received. When the very same logic runs under job management, however, and a faulty response is supplied from the dialogue table, there is no way that a re-prompt can cure the problem. In this case the bell is sounded and the error message is displayed even if the job file had suppressed dialogue, followed by the message:

```
     $17 JOB MANAGEMENT TERMINATED
```

when the statement "ACCEPT reply" is executed for the second time. Control then returns to the operator, with the type-ahead buffer cleared, so the job file can be rerun with correct input.

3.1.2 Bypassing Job Management using $$JCL

Console input obtained through the CHAR$ system routine, described in the Global Development Screen Presentation Manual, always bypasses job management. However, console input obtained through the ACCEPT statement, the ACCE$ system routine, the MAPIN statement or the PASS$ routine will only bypass job management if the system variable $$JCL has been set to the value 1. Note that $$JCL must be set to 1 before each such input operation, since the job manager will have restored its normal value of -1 by the time that the application program regains control following the affected input operation. Also remember that if the operator replies <ESCAPE> to a prompt which bypasses job management in this way, then job management will be terminated with the screen message:

```
$17 JOB MANAGEMENT TERMINATED
```

If a job file has suppressed dialogue, then all display operations issued while $$JCL is not equal to -1 will appear on the screen. This enables an explanatory display to precede a bypass prompt even in a job file with suppressed dialogue. The following example illustrates the use of this mechanism:

```
                        * $$JCL IS INITIALLY -1
                        *
DISPLAY MSG1            * PROMPT
                        * (SUPPRESSED IF DIALOGUE SUPPRESSED)
                        *
ACCEPT REPLY1           * RESPONSE OBTAINED FROM DIALOGUE
                        * TABLE IF RUNNING UNDER
                        * JOB MANAGEMENT
                        *
MOVE 1 TO $$JCL         * REQUEST BYPASS
                        *
DISPLAY MSG2            * PROMPT (NEVER SUPPRESSED)
                        *
ACCEPT REPLY2           * COLON IS DISPLAYED AND REPLY2
                        * ACCEPTED. THEN $$JCL REVERTS
                        * TO -1
                        *
DISPLAY MSG3            * PROMPT
                        * (SUPPRESSED IF DIALOGUE SUPPRESSED)
                        *
ACCEPT REPLY3           * RESPONSE OBTAINED FROM DIALOGUE
                        * TABLE IF RUNNING UNDER
                        * JOB MANAGEMENT
```

Note that the above example will also run interactively, with all three ACCEPT statements taking their input from the console.

3.2 Running a Job File under Job Management

A job file may be run under job management in exactly the same way as a program (i.e. by coding in the dialogue division of the controlling job file the name of the subject job file) in such a way that this latter name is supplied by the job manager as the response to a ready prompt.

As explained in Chapter 1, a job file is merely a special type of program which, from a job management viewpoint, operates exactly as an ordinary program during its initiation phase, when it accepts parameter values. These values may come directly from the console or, if the job file is itself running under job management, may be supplied by the job manager from the current job dialogue table. Only when a job file has received all its parameters does it establish its own dialogue table in the user stack as the current dialogue table, to

provide the subsequent dialogue for the job manager. When this table is eventually exhausted further dialogue will be taken from the most recent unexhausted job dialogue table or, if there is none, job management will be terminated and dialogue from the console will be resumed.

Thus it is possible for one job file to **nest** another, and this process may be repeated to any depth, subject only to the limitation imposed by the user area size. As each further nested job file is initiated, its dialogue table is added to the user stack, further reducing the user area available for the execution of dialogue. It is also possible for one job file to **chain** to another: in this case the dialogue table for the second job file is not established in the user stack until the dialogue table for the first job file has been exhausted and deleted from the user stack. Thus a more economical use of the user stack results when chaining is used rather than nesting.

As an example of **nesting**, suppose that a job file is to be created which will call the job file CLXMP, whose job description is given in Figure 2.1, passing it the same 3 parameters that it received from the operator console in the example dialogue of Figure 2.1.10. The following dialogue definition lines would appear in the job description of this job file:

```
...
GSM READY:CLXMP
PROGRAM NAME (SUFFIX ONLY):TEST
USER UNIT:205
XREF REQUIRED? (Y/N):Y
next statement
...
```

In this case, once CLXMP has completed execution, dialogue control will return to the dialogue table of the calling job file, at the point "next statement" above.

If, however, the call of CLXMP was followed immediately by an ENDJOB statement:

```
GSM READY:CLXMP
PROGRAM NAME (SUFFIX ONLY):TEST
USER UNIT:205
XREF REQUIRED?(Y/N):Y
ENDJOB
```

then this would be an example of **chaining**, since the dialogue table of the calling job file would exhaust itself in supplying the parameters of CXLMP. Thus by the time that the CLXMP dialogue table was established in the user stack, the calling dialogue table would already have been deleted.

```
JOB   COMP10      COMPILE UP TO 10 PROGRAMS
*
* THIS JOB COMPILES FROM 1 TO 10 PROGRAMS ON THE SAME UNIT.
*
* IT COMPILES ONE PROGRAM EACH TIME IT IS CALLED,
* THEN CALLS A FURTHER COPY OF ITSELF ITERATIVELY,
* PASSING A REDUCED SET OF PARAMETERS,
* UNTIL ALL PROGRAMS HAVE BEEN COMPILED.
*
PARAMETER DIVISION
*
      &1    X(3)  UNIT
      &2    X(6)  SOURCE FILE 1
      &3    X(6)  ? SOURCE FILE 2
      &4    X(6)  ? SOURCE FILE 3
      &5    X(6)  ? SOURCE FILE 4
      &6    X(6)  ? SOURCE FILE 5
      &7    X(6)  ? SOURCE FILE 6
      &8    X(6)  ? SOURCE FILE 7
      &9    X(6)  ? SOURCE FILE 8
      &10   X(6)  ? SOURCE FILE 9
      &11   X(6)  ? SOURCE FILE 10
*
DIALOGUE DIVISION
*
GSM READY:$COBOL
$43 SOURCE:&2 UNIT:&1
$43 COMPILATION UNIT:<CR> SIZE:<CR>
$43 COPY LIBRARY:<CR>
$43 LISTING UNIT:<CR>
$43 COMPILER OPTION:<CR>
*     CHECK IF THERE ARE MORE PROGRAMS TO COMPILE
?3
*     THERE ARE, SO RUN THE JOB FILE AGAIN
GSM READY:COMP10
*     MOVE ALL REMAINING PARAMETERS UP ONE POSITION
UNIT:&1
SOURCE FILE 1:&3
SOURCE FILE 2:&4
?4
SOURCE FILE 3:&5
?5
SOURCE FILE 4:&6
?6
SOURCE FILE 5:&7
?7
SOURCE FILE 6:&8
?8
SOURCE FILE 7:&9
?9
SOURCE FILE 8:&10
?10
SOURCE FILE 9:&11
?11
SOURCE FILE 10:<CR>
ENDJOB
```

Figure 3.2.2 - Job Description of iterative Job File

3.2.1 Programming Notes

Any jobs called from a job with suppressed dialogue will have their dialogue suppressed. However, if a job file with suppressed dialogue chains to another job file, the dialogue of the second job will not be suppressed unless that job itself specifies suppression.

When passing a parameter value between job files, care must be taken to ensure that it conforms to the format associated with it in the job description of the called job file. Failure to do this will result in the generation of one of the prompts marked with a + sign in Table

1.5, which will cause job management to be terminated. This contrasts with the acceptance of parameter values from operator input, where invalid values will cause a prompt to which the operator may reply with correct input.

3.2.2 Example - An Iterative Job File

Figure 3.2.2 shows a job file, COMP10, which accepts a variable number of parameters and invokes itself iteratively, dealing with one parameter each time that it is run, until all parameters have been processed.

The purpose of the job file is to compile from 1 to 10 source programs held on the same unit, producing compilation files also on that unit. Parameter &1 supplies the unit-id for the sequence of compilations and parameter &2, which is not conditional, supplies the name of the source file to be compiled by this execution of the job file. The remaining parameters are conditional. If source file names are supplied for parameters &3 to &11, they are accepted and placed in the dialogue table, but if <CR> is replied for any of these parameters then the job file is initiated immediately. The file defined by &2 is then compiled, and a blank separator page is output to $PR by the $P command. Then parameter &3 is checked to see if a null value was supplied and if it is null this means that all source files have been compiled and so job management is terminated. If parameter &3 is non-null, there are more compilations to be performed, so control is passed to the COMP10 job file again, passing the same unit number, but cycling the source file parameters so that parameter &3 will be treated as &2, and so on.

Note that after each parameter is supplied to the next iteration, by a dialogue line such as:

```
SOURCE FILE 2:&4
```

It is tested to see whether it was initially specified by a line such as:

```
?4
```

This test allows the job manager to free the user stack space occupied by the previous iteration of COMP10 as soon as it is no longer required. If the test were omitted n copies of COMP10 would be present by the time the n'th compilation took place, possibly causing the job to fail if used for a sufficiently large number of source files. By including the test immediately after the parameter is supplied we ensure that a maximum of two copies (of the invoking and invoked job) are present in the stack at any one time.

COMP10 is of course reloaded from direct access storage every time it is invoked, so it should ideally be kept online during the entire process, or you will have to mount it when requested by a prompt which bypasses job management.

The following screen dialogue is produced when COMP10 is used to compile two programs, TEST1 and TEST2, on unit 101:

```
GSM READY:COMP10

COMPILE UP TO 10 PROGRAMS
```

```
UNIT:205
SOURCE FILE:TEST1
SOURCE FILE:TEST2
SOURCE FILE:<CR>
GSM READY:$COBOL
$43 SOURCE:TEST1 UNIT:205
$43 COMPILATION UNIT: SIZE:
$43 COPY LIBRARY:
$43 LISTING UNIT:
$43 COMPILER OPTION:
$43 COMPILING
$43 END OF FIRST PASS
$43 NUMBER OF ERRORS     0
$43 NUMBER OF WARNINGS 0
$43 COMPILATION COMPLETED
GSM READY:COMP10

COMPILE UP TO 10 PROGRAMS

UNIT:205
SOURCE FILE:TEST2
SOURCE FILE:
GSM READY:$COBOL
$43 SOURCE:TEST2 UNIT:205
$43 COMPILATION UNIT: SIZE:
$43 COPY LIBRARY:
$43 LISTING UNIT:
$43 COMPILER OPTION:
$43 COMPILING
$43 END OF FIRST PASS
$43 NUMBER OF ERRORS     0
$43 NUMBER OF WARNINGS 0
$43 COMPILATION COMPLETED
GSM READY:
```

3.3 Invoking a Job File from an Application Program

Since a job file is a program file, it may be invoked from an application program by a RUN (or CHAIN) statement. For example:

```
RUN "CLXMP"
```

will cause job file CLXMP (Figure 2.1) to gain control and to prompt for its parameters on the screen.

Note that it is not possible to pass parameters directly from the calling program to the job file, but the applications work area, $$AREA, may be used to pass a small number of values to a subject program.

If any error occurs during the loading of the job file (because it is not present on the program residence device, for example) then the calling program will be terminated in error.

3.3.1 Programming Note

Each subject program of the invoked job file, including the first, is run from a ready prompt, with the job manager supplying the name of the program to be run in response to each ready prompt. At each ready prompt the monitor performs end-of-job processing, and this ensures that each subject program starts execution in an initial state (i.e. with the console in teletype mode) with the ESCAPE key enabled, with no work space allocated and without a user-specified end-of-job routine. Since a job file is normally run from a ready prompt, it too expects to be run from this same initial state. This will be the case if you invoke the job file using the RUN statement. If you use CHAIN (only recommended if you require your programs to run under an earlier

version of Global System Manager which does not support the RUN statement) you must establish the initial state yourself, by:

- resetting teletype mode;
- enabling the ESCAPE key;
- releasing work space;
- detaching any end-of-job routine.

The following code shows how the initial state might be re-established before chaining to the example job file:

```
SCROLL                  * RESET TELETYPE MODE
MOVE 0 TO $$ESC         * ENABLE ESCAPE KEY
CALL FREE$              * RELEASE WORKSPACE
CALL EOJ$               * DETACH END-OF-JOB ROUTINE
*
CHAIN "CLXMP"           * INVOKE JOB FILE CLXMP
```

3.4 Producing a Tailored Job Initiator using JOB$

In some situations the standard job file as created by the $JOB command may not provide sufficient flexibility, and for this reason the JOB$ system routine is provided so that you can develop a **tailored job initiator**.

A tailored job initiator performs a similar function to a job file created by $JOB, inasmuch as both build dialogue tables on the user stack and then call the job manager to initiate the dialogue. However, it is the method used to build the dialogue table which distinguishes a tailored job initiator from a job file. For a job file, the contents of its dialogue table are broadly determined when it is created by $JOB, although some flexibility can be provided by parameter values which you supply when the job file is executed. With a tailored job initiator however, the dialogue table is assembled under program control and the variety of possible dialogue tables is limited only by the sophistication of the initiator.

A typical initiator would be designed to generate dialogue to perform a particular function, and would be capable of generating every variant of dialogue table that might be needed to achieve that function. When it is executed, the initiator will obtain parameter information (e.g. by prompting for it) and will generate the required dialogue table accordingly. This it does by passing **dialogue packets** in successive calls to the JOB$ routine which actually builds the dialogue table on the user stack. When all dialogue has been built, a special dialogue packet causes the complete dialogue to be initiated.

3.4.1 The JOB$ Routine

The JOB$ routine is invoked by a call of the form:

```
CALL JOB$ USING dialogue-packet
```

where dialogue-packet is a character variable or character literal containing job dialogue. Each dialogue packet terminates with an exclamation mark which is not itself part of the job dialogue, but merely enables JOB$ to detect the end of the packet.

During the execution of a typical job initiator, the dialogue packets passed in sequence to JOB$ correspond to a subset of the job description dialogue division (see 2.1.5 - 2.1.9). There are six basic types of dialogue packet that may be passed to JOB$, as follows:

CALL JOB$ USING "DIALOGUE DIVISION!"

or:

CALL JOB$ USING "DIALOGUE DIVISION (SUPPRESS)!"

These dialogue packets correspond to the dialogue division header described in 2.1.5. The first packet passed by a job initiator to JOB$ must be of this form, since it causes JOB$ to initialise its dialogue table prior to building it.

CALL JOB$ USING "text!"

Here text corresponds broadly to the text which can appear in a dialogue definition line (see 2.1.6). The text consists of one or more characters, including spaces if required but excluding the colon, minus, plus and exclamation mark characters. Additionally it must not contain the string ENDJOB. This packet is supported for documentation purposes only, and is completely ignored by JOB$.

CALL JOB$ USING ":response !"

Here response is as described in 2.1.6, except that the space character is the only acceptable terminator and the special parameter name response introduced by an ampersand is not allowed. For each packet of this type, JOB$ adds the response to the dialogue table that it is building.

CALL JOB$ USING "-prompt-!"

Here prompt is the text of a pause prompt as described in 2.1.7. For each packet of this type, JOB$ adds the pause prompt to the dialogue table that it is building.

CALL JOB$ USING "+message+!"

Here message is the text of a message as described in 2.1.8. For each packet of this type, JOB$ adds the message to the dialogue table that it is building.

CALL JOB$ USING "ENDJOB!"

When JOB$ receives this packet it causes the dialogue table to be initiated. This must be the last packet passed by the initiator to JOB$. Control is not returned to the initiator from this call of JOB$.

It is possible to concatenate the basic dialogue packets described above to produce more complex packets. In this case only the last basic packet is terminated by an exclamation mark. This technique reduces the number of calls on JOB$ needed to build a complete dialogue table. For example, the following call of JOB$ will build and initiate the dialogue table needed to run the $P command which throws a blank page on the system printer:

```
CALL JOB$ USING "DIALOGUE DIVISION:$P ENDJOB!"
```

3.4.2 Programming Notes

When coding a job initiator it is extremely important to check that each dialogue packet is terminated by an exclamation mark character. If this is omitted, additional characters will be unintentionally passed to JOB$, with unpredictable results. Also each response within a dialogue packet must be terminated with a space character, each pause prompt with a minus character and each message with a plus character. In particular, if the last item in a packet is a response the packet must end with a space followed by an exclamation mark. By following these simple rules the possibilities of errors will be reduced, but when errors do occur they will produce stop codes documented in the Global Utilities Manual.

It is possible to run a tailored job initiator under job management in exactly the same way as a job file is run under job management (see 3.2). Remember that the job dialogue table built by the initiator is not established as the current job dialogue table until the "ENDJOB" dialogue packet is passed to JOB$. Until this time the initiator behaves like a job file in its parameter-acceptance phase.

3.4.3 The Example Job Initiator, CLX

Appendix A contains the compilation listing of an example job initiator, CLX. This initiator is capable of building the dialogue table needed to perform any combination of compile, link and cross-reference steps upon a named program. Thus it performs a superset of the functions possible with the example job file, CLXMP, given in Figure 2.1.

You execute the job initiator by typing its name in response to a ready prompt in the normal way, and it displays its title and prompts for the program name, user unit and copy library name:

```
GSM READY:CLX

COMPILE, LINK AND XREF JOB INITIATOR

PROGRAM NAME (SUFFIX ONLY):TEST
USER UNIT:205
COPY LIBRARY (DEFAULT = NONE):TT
COMPILATION REQUIRED? (DEFAULT = Y):
```

You are then asked if the program is to be compiled, and if you reply Y or <CR> the necessary dialogue to achieve the compilation will be passed to JOB$. Note that three separate dialogue packets are used to enable different dialogue to be selected depending on the presence or absence of a copy library. A standard job file cannot provide this facility as it can only support a fixed sequence of responses. Note also that the dialogue packets are parameterised with program name, unit name and copy library name before being passed to JOB$. This parameterisation is equivalent to the ampersand parameters supported by a job file and explains why there is no need for ampersand parameters in JOB$ dialogue. If you reply N to this prompt, no compilation dialogue will be generated.

```
COMPILATION REQUIRED? (DEFAULT = Y):<CR>
X-REFERENCE REQUIRED? (DEFAULT = Y):
```

You are now asked if the program is to be cross-referenced:

```
X-REFERENCE REQUIRED? (DEFAULT = Y):<CR>
```

```
LINK REQUIRED? (DEFAULT = Y):
```

The initiator action is similar to that described above for the compilation.

Finally you are asked if the compilation file is to be linked. If you reply Y or <CR> the dialogue table for a linkage edit will be generated, but an additional prompt appears asking for the name of any extra compilation files or libraries to be included in the linkage edit:

```
LINK REQUIRED? (DEFAULT = Y):<CR>
ADDITIONAL LINK MODULE NAME (DEFAULT = NONE):
```

Each time that you supply the name of an additional link module the necessary dialogue packet will be constructed and passed to JOB$. The same prompt will then be redisplayed. Once you reply <CR> to this prompt the final linkage edit dialogue packet will be passed to JOB$:

```
ADDITIONAL LINK MODULE NAME (DEFAULT = NONE):MOD1
ADDITIONAL LINK MODULE NAME (DEFAULT = NONE):MOD2
ADDITIONAL LINK MODULE NAME (DEFAULT = NONE):<CR>
```

The initiator then generates dialogue to throw a clean page on the printer and finally initiates the generated dialogue by calling JOB$ with the special "ENDJOB" parameter. The generated dialogue now appears at the console as the dialogue table supplies input in place of the console:

```
GSM READY:

JOB INITIATED

GSM READY:$COBOL
.........
......... (rest of dialogue)
.........
GSM READY:
```

The source, S.CLX, of this initiator is supplied as part of any advanced Global Cobol development system so that you can compile and link it to produce the CLX job initiator to use in your own program development, or indeed modify it so that it exactly meets your own special requirements. The S.CLX file is installed on the system volume containing the Global Cobol compiler.

Appendix A - Example Listings

The following pages contain three job listings produced by $JOB from example job descriptions, and a compilation listing of an example tailored job initiator. In the job listings, the first column numbers the source line and the second column gives the number of bytes of dialogue table generated by each line. The compilation listing format is described in MU-A.

L.CLXMP is the job listing for the example job description given in Figure 2.1.

L.COMP10 is the job listing for the job description of the iterative job file given in Figure 3.2.2, and shows how a job file may invoke a further copy of itself, passing on a reduced parameter set.

L.INSPR is the job listing for a job description which shows how an application product may be installed from distribution diskettes onto operational volumes. This example has been deliberately simplified to highlight the use of the volume-id checking facilities of $F and $LIB.

L.CLX is the compilation listing of the example job initiator which is described in 3.4.3.

```
JOB LIST OF CLXMP COMPILE, LINK & XREF MAIN PROG 14/4/88 11.45.06 PAGE 1
1     0     JOB CLXMP COMPILE, LINK & XREF MAIN PROGRAM
2     0     *
3     0     * THIS JOB DESCRIPTION COMPILES THE SOURCE OF A MAIN PROGRAM,
4     0     * LINKAGE EDITS IT WITH THE SYSTEM LIBRARIES,
5     0     * AND OPTIONALLY PRODUCES A CROSS-REFERENCE LISTING.
6     0     * ALL LISTINGS ARE PRODUCED ON THE LOGICAL PRINTER, $PR.
7     0     *
8     0     PARAMETER DIVISION
9     0     *
10    8           &1 X(6) PROGRAM NAME (SUFFIX ONLY)
11    5           &2 X(3) USER UNIT
12    3           &3 X(1) ? XREF REQUIRED? (Y/N)
13    0     *
14    0     DIALOGUE DIVISION
15    0     *
16    0     * FIRSTLY, COMPILE THE PROGRAM
17    24    +COMPILATION STARTING+
18    1     GSM READY:<CR>
19    9     GSM READY:$COBOL
20    4     $43 SOURCE:&1 UNIT:&2
21    2     $43 COMPILATION UNIT:<CR> SIZE:<CR>
22    1     $43 COPY LIBRARY:<CR>
23    1     $43 LISTING UNIT:<CR>
24    1     $43 COMPILER OPTION:<CR>
25    0     *
26    0     *
27    0     * THEN LINK THE PROGRAM.
28    28    +LINKAGE EDITING STARTING+
29    1     GSM READY:<CR>
30    8     GSM READY:$LINK
31    4     $44 LINK:&1 UNIT:&2
32    1     $44 LINK:<CR>
33    2     $44 PROGRAM:<CR> UNIT:<CR>
34    1     $44 LISTING UNIT:<CR>
35    1     $44 LINK OPTION:<CR>
36    0     *
37    2     *
38    0     * CHECK IF A CROSS-REFERENCE HAS BEEN SELECTED.
39    1     ?3
40    0     * PAUSE TO ALLOW $XREF TO BE MOUNTED.
41    0     * NOTE: THIS PAUSE PROMPT IS NOT STRICTLY NECESSARY,SINCE SYSTEM
42    0     * MANAGER PRODUCES OWN MOUNT PROMPT IF $XREF IS NOT FOUND.
43    42    -CONFIRM XREF REQUIRED, <ESC> OTHERWISE-
44    0     * NOW PRODUCE THE CROSS-REFERENCE.
45    1     GSM READY:<CR>
46    8     GSM READY:$XREF
47    4     $54 SOURCE:&1 UNIT:&2
48    1     $54 COPY LIBRARY:<CR>
49    1     $54 LISTING UNIT:<CR>
50    1     $54 X-REFERENCE OPTION:<CR>
51    0     * FINALLY, THROW ANOTHER BLANK SEPARATOR PAGE, THEN STOP.
52    0     *
53    0     *
54    0     ENDJOB
```

Job Listing, L.CLXMP - Page 1

```
JOB LIST OF CLXMP COMPILE, LINK & XREF MAIN PROG 14/4/88 11.45.07 PAGE 2


END OF JOB DESCRIPTION -- 164 BYTES OF JOB DIALOGUE TABLE USED

NUMBER OF WARNINGS 0
NUMBER OF ERRORS 0

11318 (#2B82) BYTES OF USER AREA REQUIRED TO INITIATE THIS JOB
1210 (#04BA) BYTES OF USER STACK WILL BE USED THROUGHOUT THE EXECUTION OF THIS JOB

SOURCE FILE S.CLXMP OF SIZE    1.4K BYTES USED ON UNIT 205
JOB FILE CLXMP OF SIZE   3.0K BYTES PRODUCED ON UNIT 205
LISTING FILE L.CLXMP OF SIZE   5.9K BYTES PRODUCED ON UNIT 206
PRODUCED BY JOB MANAGEMENT V8.1
```

Job Listing, L.CLXMP - Page 2

```
JOB LIST OF COMP10 COMPILE UP TO 10 PROGRAMS     14/04/88 11.46.49 PAGE 1

1     0     JOB COMP10 COMPILE UP TO 10 PROGRAMS
2     0     *
3     0     * THIS JOB COMPILES FROM 1 TO 10 PROGRAMS ON THE SAME UNIT.
4     0     *
5     0     * IT COMPILES ONE PROGRAM EACH TIME IT IS CALLED,
6     0     * THEN CALLS A FURTHER COPY OF ITSELF ITERATIVELY,
7     0     * PASSING A REDUCED SET OF PARAMETERS,
8     0     * UNTIL ALL PROGRAMS HAVE BEEN COMPILED.
9     0     *
10    0     PARAMETER DIVISION
11    0     *
12    5           &1    X(3)  UNIT
13    8           &2    X(6)  SOURCE FILE 1
14    8           &3    X(6)  ? SOURCE FILE 2
15    8           &4    X(6)  ? SOURCE FILE 3
16    8           &5    X(6)  ? SOURCE FILE 4
17    8           &6    X(6)  ? SOURCE FILE 5
18    8           &7    X(6)  ? SOURCE FILE 6
19    8           &8    X(6)  ? SOURCE FILE 7
20    8           &9    X(6)  ? SOURCE FILE 8
21    8           &10   X(6)  ? SOURCE FILE 9
22    8           &11   X(6)  ? SOURCE FILE 10
23    0     *
24    0     DIALOGUE DIVISION
25    0     *
26    9     GSM READY:$COBOL
27    4     $43 SOURCE:&2 UNIT:&1
28    2     $43 COMPILATION UNIT:<CR> SIZE:<CR>
29    1     $43 COPY LIBRARY:<CR>
30    1     $43 LISTING UNIT:<CR>
31    1     $43 COMPILER OPTION:<CR>
32    0     *
33    0     *
34    0     * CHECK IF THERE ARE MORE PROGRAMS TO COMPILE
35    1     ?3
36    0     * THERE ARE, SO RUN THE JOB FILE AGAIN
37    9     GSM READY:COMP10
38    0     * MOVE ALL REMAINING PARAMETERS UP ONE POSITION
39    2     UNIT:&1
40    2     SOURCE FILE 1:&3
41    2     SOURCE FILE 2:&4
42    1     ?4
43    2     SOURCE FILE 3:&5
44    1     ?5
45    2     SOURCE FILE 4:&6
46    1     ?6
47    2     SOURCE FILE 5:&7
48    1     ?7
49    2     SOURCE FILE 6:&8
50    1     ?8
51    2     SOURCE FILE 7:&9
52    1     ?9
53    2     SOURCE FILE 8:&10
54    1     ?10
55    2     SOURCE FILE 9:&11
56    1     ?11
57    1     SOURCE FILE 10:<CR>
58    0     ENDJOB
```

Job Listing, L.COMP10 - Page 1

```
JOB LIST OF COMP10 COMPILE UP TO 10 PROGRAMS     14/04/88 11.46.49 PAGE 2


END OF JOB DESCRIPTION -- 142 BYTES OF JOB DIALOGUE TABLE USED

NUMBER OF WARNINGS 0
NUMBER OF ERRORS 0

11094 (#2B56) BYTES OF USER AREA REQUIRED TO INITIATE THIS JOB
 1188 (#04A4) BYTES OF USER STACK WILL BE USED THROUGHOUT THE EXECUTION OF THIS JOB

SOURCE FILE S.COMP10 OF SIZE   1.1K BYTES USED ON UNIT 205
JOB FILE COMP10 OF SIZE  3.2K BYTES PRODUCED ON UNIT 205
LISTING FILE L.COMP10 OF SIZE  6.2K BYTES PRODUCED ON UNIT 206
PRODUCED BY JOB MANAGEMENT V8.1
```

Job Listing, L.COMP10 - Page 2

```
JOB LIST OF INSPR INSTALL PAYROLL ONTO 1 VOLUME 14/04/88 11.48.59 PAGE 1

 1    0     JOB INSPR INSTALL PAYROLL ONTO 1 VOLUME
 2    0     *
 3    0     * THIS JOB FILE ILLUSTRATES THE USE OF THE VOLUME-ID CHECKING
 4    0     * FACILITIES OF $F AND $LIB.
 5    0     *
 6    0     * IT INSTALLS THE PROGRAM LIBRARY OF A PAYROLL SYSTEM BY MERGING THE
 7    0     * TWO LIBRARIES, P.PRA AND P.PRB, DISTRIBUTED ON VOLUMES PRA AND PRB
 8    0     * RESPECTIVELY, INTO A SINGLE LIBRARY, P.PR, HELD ON VOLUME PRGRES.
 9    0     * THE MENU PROGRAM, $MENU, IS THEN COPIED FROM SYSRES TO PRGRES.
10    0     * THE JOB FILE ITSELF IS LOADED FROM VOLUME P.PRA ON UNIT $P.
 11   0     *
 12   0     PARAMETER DIVISION
 13   0     *
 14   5           &1 X(3) OUTPUT UNIT (PRGRES)
 15   0     *
 16   0     DIALOGUE DIVISION
 17   0     *
 18   7     GSM READY:$LIB
 19   10    $95 TARGET LIBRARY:<CTRL C> VOLUME-ID:PRGRES
 20   9     $95 TARGET LIBRARY:P.PR UNIT:&1
 21   5     $95 NEW?:Y SIZE:<CR>
 22   1     $95 TITLE:<CR>
 23   0     $95 LIBRARY MAINTENANCE
 24   7     :MER FROM UNIT:<CR>
 25   9     $95 FILE:P.PRA MEMBER:<CTRL B>
 26   0           (LIST OF MEMBERS OF P.PRA AS THEY ARE MERGED)
 27   0     $95 LIBRARY MAINTENANCE
 28   10    :I :PRB
 29   0     $95 LIBRARY MAINTENANCE
 30   3     :MER FROM UNIT:<CR>
 31   9     $95 FILE:P.PRB MEMBER:<CTRL B>
 32   0           (LIST OF MEMBERS OF P.PRB AS THEY ARE MERGED)
 33   0     $95 LIBRARY MAINTENANCE
 34   7     :TRU NEW SPACE SPACE:<CR> TRUNCATED
 35   0     $95 LIBRARY MAINTENANCE
 36   6     :END
 37   1     $95 TARGET LIBRARY:<ESCAPE>
 38   5     GSM READY:$F
 39   6     $66 INPUT DEVICE:$CP
 40   2     $66 OUTPUT DEVICE:&1
 41   0     $66 FILE MAINTENANCE
 42   11    :I :SYSRES
 43   0     $66 FILE MAINTENANCE
 44   6     :O :PRGRES
 45   0     $66 FILE MAINTENANCE
 46   16    :COP :$MENU TO:<CR> SIZE:<CR>
 47   0     $66 FILE MAINTENANCE
 48   1     :<ESCAPE>
 49   0     *
 50   0     ENDJOB
```

Job Listing, L.INSPR - Page 1

```
JOB LIST OF INSPR INSTALL PAYROLL ONTO 1 VOLUME 14/04/88 11.48.59 PAGE 2


END OF JOB DESCRIPTION -- 136 BYTES OF JOB DIALOGUE TABLE USED

NUMBER OF WARNINGS 0
NUMBER OF ERRORS 0

10860 (#2A6C) BYTES OF USER AREA REQUIRED TO INITIATE THIS JOB
 1088 (#0440) BYTES OF USER STACK WILL BE USED THROUGHOUT THE EXECUTION OF THIS JOB

SOURCE FILE S.INSPR OF SIZE    1.3K BYTES USED ON UNIT 205
JOB FILE INSPR OF SIZE   2.8K BYTES PRODUCED ON UNIT 205
LISTING FILE L.INSPR OF SIZE   5.5K BYTES PRODUCED ON UNIT 206
PRODUCED BY JOB MANAGEMENT V8.1
```

Job Listing, L.INSPR - Page 2

```
LISTING OF CLX COMPILE, LINK & XREF SEQUENCER 14/04/88 11.55.22 PAGE 1

 2                PROGRAM CLX
 3                *
 4                * THIS PROGRAM ILLUSTRATES HOW THE JOB$ SYSTEM ROUTINE MAY BE USED
 5                * TO PRODUCE A TAILORED JOB BUILDER/INITIATOR.
 6                *
 7                * WITH A MINIMUM OF OPERATOR INTERACTION, A JOB IS CREATED TO
 8                * COMPILE, CROSS-REFERENCE AND LINK A SELECTED PROGRAM. ANY OF THE
 9                * THREE DEVELOPMENT STEPS MAY BE OMITTED.
10                *
11   0000         DATA DIVISION
12                *
13   0000         01    C-PKTS                        * $COBOL JOB DIALOGUE PACKETS
14   0000          02   C-PKT1                        * $COBOL JOB DIALOGUE PACKET 1
15   0000          03   FILLER      PIC X(?)
16   0000                     VALUE "GSM READY:$COBOL "
17   0011                     VALUE "$43 SOURCE:"
18   001C          03   CNAME PIC X(6)           * SOURCE FILE NAME
19   0022          03   FILLER      PIC X(?)
20   0022                     VALUE " UNIT:"
21   0028          03   CUNIT PIC X(3)           * SOURCE FILE UNIT
22   002B          03   FILLER      PIC X(?)
23   002B                     VALUE " $43 COMPILATION UNIT:"
24   0041          03   CCOMPU      PIC X(3)           * COMPILATION FILE UNIT
25   0044          03   FILLER      PIC X(?)
26   0044                     VALUE " SIZE:<CR> "
27   0051                     VALUE "$43 COPY LIBRARY:"
28   0062          03   CCOP  PIC X(6)           * COPY LIBRARY NAME
29   0068          03   FILLER      PIC X(?)
30   0068                          VALUE " !"
31   006A          02   C-PKT2      PIC X(?)           * $COBOL JOB DIALOGUE PACKET 2
32   006A                     VALUE " UNIT:<CR> !"
33   0078          02   C-PKT3      PIC X(?)           * $COBOL JOB DIALOGUE PACKET 3
34   0078                     VALUE "$43 LISTING UNIT:$PR "
35   008D                     VALUE "$43 COMPILER OPTION:<CR> !"
36         *
37   00A9         01    X-PKTS                        * $XREF JOB DIALOGUE PACKETS
38   00A9          02   X-PKT1                        * $XREF JOB DIALOGUE PACKET 1
39   00A9          03   FILLER      PIC X(?)
40   00A9                     VALUE "GSM READY:$XREF "
41   00B9                     VALUE "$54 SOURCE:"
42   00C4          03   XNAME PIC X(6)           * SOURCE FILE NAME
43   00CA          03   FILLER      PIC X(?)
44   00CA                          VALUE " UNIT:"
45   00D0          03   XUNIT PIC X(3)           * SOURCE FILE UNIT
46   00D3          03   FILLER      PIC X(?)
47   00D3                     VALUE " $54 COPY LIBRARY:"
48   00E5          03   XCOP  PIC X(6)           * COPY LIBRARY NAME
49   00EB          03   FILLER      PIC X(?)
50   00EB                          VALUE " !"
51   00ED          02   X-PKT2      PIC X(?)           * $XREF JOB DIALOGUE PACKET 2
52   00ED                          VALUE " UNIT:<CR> !"
53   00FB          02   X-PKT3      PIC X(?)           * $XREF JOB DIALOGUE PACKET 3
54   00FB                     VALUE "$54 LISTING UNIT:$PR "
55   0110                     VALUE "$54 X REFERENCE OPTION:<CR> !"
```

Compilation Listing, L.CLX - Page 1

```
LISTING OF CLX COMPILE, LINK & XREF SEQUENCER 14/04/88 11.55.22 PAGE 2

 57   012F        01    L-PKTS                          * $LINK JOB DIALOGUE PACKETS
 58   012F         02   L-PKT1                          * $LINK JOB DIALOGUE PACKET 1
 59   012F         03   FILLER      PIC X(?)
 60   012F                    VALUE "GSM READY:$LINK "
 61   013F                    VALUE "$44 LINK:"
 62   0148         03   LNAME1      PIC X(6)           * FIRST COMPILATION FILE NAME
 63   014E         03   FILLER      PIC X(?)
 64   014E                    VALUE " UNIT:"
 65   0154         03   LUNIT PIC X(3)          * COMPILATION FILE UNIT
 66   0157         03   FILLER      PIC X(?)
 67   0157                    VALUE " !"
 68              *
 69   0159         02   L-PKT2                          * $LINK JOB DIALOGUE PACKET 2
 70   0159         03   FILLER      PIC X(?)
 71   0159                    VALUE "$44 LINK:"
 72   0162         03   LNAME2      PIC X(6)           * NEXT COMPILATION FILE NAME
 73   0168         03   FILLER      PIC X(?)
 74   0168                         VALUE " UNIT:<CR> !"
 75              *
 76   0176         02   L-PKT3                          * $LINK JOB DIALOGUE PACKET 3
 77   0176         03   FILLER      PIC X(?)
 78   0176                    VALUE "$44 LINK:<CR> "
 79   0186                    VALUE "$44 PROGRAM:"
 80   0192         03   LPROG PIC X(6)          * PROGRAM FILE NAME
 81   0198         03   FILLER      PIC X(?)
 82   0198                         VALUE " UNIT:<CR> "
 83   01A5                    VALUE "$44 LISTING UNIT:$PR "
 84   01B6                    VALUE "$44 LINK OPTION:<CR> !"
 85              *
 86   01CE        77    REPLY PIC X
 87   01CF        77    NAME  PIC X(6)
 88   01D5        77    UNIT  PIC X(3)
 89   01D8        77    COPLIB      PIC X(6)
```

Compilation Listing, L.CLX - Page 2

```
LISTING OF CLX COMPILE, LINK & XREF SEQUENCER 14/04/88 11.55.22 PAGE 3

 91   01DE 0 0    PROCEDURE DIVISION
 92   01DE 0 0    SECTION MAIN
 93               *
 94               * DISPLAY JOB TITLE BEFORE ACCEPTING PARAMETERS
 95               *
 96   01E4 0 0         DISPLAY SPACE                * DISPLAY BLANK LINE
 97   01E6 0 0         DISPLAY "COMPILE, LINK AND XREF SEQUENCER"
 98   0208 0 0         DISPLAY SPACE                * DISPLAY BLANK LINE
 99         *
100   020A 0 0         CALL JOB$ USING "DIALOGUE DIVISION!" * START BUILDING JOB
101              *
102              *  GENERATE  DIALOGUE  FOR  MESSAGE  TO  BE  DISPLAYED  WHEN  JOB  IS
INITIATED
103              *
104   0222 0 0         CALL JOB$ USING "GSM READY!"  * COMMENT DIALOGUE
105   0232 0 0         CALL JOB$ USING "+ ++JOB INITIATED+!" * NOTE: TWO MESSAGES
106   024C 0 0         CALL JOB$ USING ":<CR> !" * PRODUCE "GSM READY:" DISPLAY
107              *
108              * ACCEPT PARAMETER VALUES FOR THIS INITIATION OF THE JOB
109              *
110   025C 0 0         DISPLAY "PROGRAM NAME (SUFFIX ONLY)"
111   0278 0 0         ACCEPT NAME
112   0284 0 0         DISPLAY "USER UNIT"
113   0290 0 0         ACCEPT UNIT
114   029C 0 0         DISPLAY "COPY LIBRARY (DEFAULT = NONE)"
115   02BC 0 0         ACCEPT COPLIB NULL
116   02CC 0 1              MOVE "<CR>" TO COPLIB * NO COPY LIBRARY
117   02D8 0 0         END
118              *
119              * GENERATE $COBOL JOB DIALOGUE PACKETS IF REQUESTED
120              *
121   02D8 0 0         DISPLAY "COMPILATION REQUIRED? (DEFAULT = Y)"
122   02FE 0 0         ACCEPT REPLY NULL
123   030E 0 0         OR REPLY = "Y"
124   0318 0 1              MOVE NAME TO CNAME
125   0320 0 1              MOVE UNIT TO CUNIT
126   0328 0 1              MOVE UNIT TO CCOMPU
127   0330 0 1              MOVE COPLIB TO CCOP
128   0338 0 1              CALL JOB$ USING C-PKT1  * $COBOL JOB PACKET PART 1
129   0340 0 1              IF COPLIB NOT = "<CR>" * COPY LIBRARY REQUIRED
130   0350 0 2                   CALL JOB$ USING C-PKT2
131   0358 0 1              END
132   0358 0 1              CALL JOB$ USING C-PKT3  * $COBOL JOB PACKET PART 3
133   0360 0 0              END
134              *
135              * GENERATE $XREF JOB DIALOGUE PACKETS IF REQUESTED
136              *
137   0360 0 0         DISPLAY "X-REFERENCE REQUIRED? (DEFAULT = Y)"
138   0386 0 0         ACCEPT REPLY NULL
139   0396 0 0         OR REPLY = "Y"
140   03A0 0 1              MOVE NAME TO XNAME
141   03A8 0 1              MOVE UNIT TO XUNIT
142   03B0 0 1              MOVE COPLIB TO XCOP
143   03B8 0 1              CALL JOB$ USING X-PKT1  * $XREF JOB PACKET PART 1
144   03C0 0 1              IF COPLIB NOT = "<CR>" * COPY LIBRARY REQUIRED
145   03D0 0 2                   CALL JOB$ USING X-PKT2
146   03D8 0 1              END
147   03D8 0 1              CALL JOB$ USING X-PKT3  * $XREF JOB PACKET PART 3
148   03E0 0 0         END
```

Compilation Listing, L.CLX - Page 3

```
LISTING OF CLX COMPILE, LINK & XREF SEQUENCER 14/04/88 11.55.22 PAGE 4

150                 *
151                 * GENERATE $LINK JOB DIALOGUE PACKETS IF REQUESTED
152                 *
153   03E0 0 0            DISPLAY "LINK REQUIRED? (DEFAULT = Y)"
154   03FE 0 0            ACCEPT REPLY NULL
155   040E 0 0            OR REPLY = "Y"
156   0418 0 1                  MOVE NAME TO LNAME1
157   0420 0 1                  MOVE NAME TO LPROG
158   0428 0 1                  MOVE UNIT TO LUNIT
159   0430 0 1                  CALL JOB$ USING L-PKT1  * $LINK JOB PACKET PART 1
160   0438 0 1    ADDMOD.
161   0438 0 1                  DISPLAY "ADDITIONAL LINK MODULE NAME (DEFAULT = NONE)"
162   0466 0 1                  ACCEPT LNAME2 NULL GO TO CONT
163   0476 0 1                  CALL JOB$ USING L-PKT2  * $LINK JOB PACKET PART 2
164   047E 0 1                  GO TO ADDMOD
165   0482 0 1    CONT.
166   0482 0 1                  CALL JOB$ USING L-PKT3  * $LINK JOB PACKET PART 3
167   048A 0 0            END
168                 *
169                 * GENERATE DIALOGUE TO THROW A BLANK PAGE
170                 *
171   048A 0 0            CALL JOB$ USING ":$P !"       * THROW PAGE
172                 *
173                 * GENERATE DIALOGUE TO INITIATE THE JOB
174                 *
175   0496 0 0            CALL JOB$ USING "ENDJOB!"     * INITIATE JOB
176                 *
177                 * THE STOP RUN STATEMENT IS INCLUDED FOR TIDYNESS - JOB$ DOES NOT
178                 * RETURN CONTROL TO THE INITIATOR AFTER RECEIVING "ENDJOB!"
179                 *
180   04A4 0 0            STOP RUN
181                 *
182   04A8 0 0    ENDPROG
```

Compilation Listing, L.CLX - Page 4

```
LISTING OF CLX STATISTICS            14/04/88 11.55.22 PAGE 5

NUMBER OF ERRORS 0
NUMBER OF WARNINGS 0

COMPILATION OPTIONS IN FORCE :

TR - TRACE INFORMATION GENERATED IN COMPILATION FILE
NLN - NO LONG NAMES, THE FIRST 6 CHARACTERS ARE SIGNIFICANT
SD - SYMBOLIC DEBUG RECORD GENERATED IN COMPILATION FILE

PROGRAM SIZE = 04A8 BYTES (HEXADECIMAL)

TOTAL NUMBER OF LINES    182 (EXCLUDING COMMENTS 135)

SOURCE FILE -     S.CLX  ON 114 CREATED 14/04/88
COMPILATION -     C.CLX  ON 114 SIZE       1.8K
LISTING FILE -    L.CLX  ON 500 SIZE       8.7K

MACHINE - IBM PS/2-95
VERSION - V8.1

COMPILATION COMPLETED
```

Compilation Listing, L.CLX - Page 5

Appendix B - Included Routines

System routines referenced by the CALL statement are included in your program from the system library when it is linkage edited. If you create a tailored initiator using the JOB$ system routine, the 2 members of C.$APF shown in the table below are included. The SIZE column indicates the approximate memory size taken, rounded up to the nearest .1K (K=1024).

Global Cobol statement	Program name of the subroutine included	Size (Kb)
CALL JOB$	CX$A, OJ$A	5.2

Table B - Included Routines

Appendix C – Job Listing Error and Warning Messages

This appendix describes the error and warning messages produced by the Job File Builder, $JOB, in the listing file. If an **error** occurs, this indicates a serious fault and no job file will have been created. $JOB is able to recover following a **warning** however, and the description of the warning message specifies the recovery action taken. Job files subject to warnings are executable, but it is of course good practice to correct the job description so that a clean job listing can be obtained.

Each message consists of the error or warning number of one or two digits and a short explanation. In general the message follows the line in error, but where a missing line is referred to the message is printed where the missing line should have been.

*** WARNING 1 LINE NOT RECOGNISED**

An invalid line has been found where a PARAMETER DIVISION, DIALOGUE DIVISION or comment line only would be valid. The invalid line is ignored.

*** WARNING 2 FIRST LINE IS NOT A JOB LINE**

The first line of a job description must be a JOB line. The invalid line is ignored.

*** WARNING 3 PARAMETER DIVISION STATEMENT NOT PRESENT**

A parameter definition line has been found before a PARAMETER DIVISION line has been processed. The PARAMETER DIVISION line is assumed to have occurred immediately before the current line which is then processed accordingly.

*** WARNING 4 DIALOGUE DIVISION STATEMENT NOT PRESENT**

A dialogue line, pause line, message line, conditional exit line or ENDJOB line has been found before a DIALOGUE DIVISION line has been processed. The DIALOGUE DIVISION line is assumed to have occurred immediately before the current line which is then processed accordingly.

*** WARNING 5 ENDJOB NOT PRESENT BEFORE END OF FILE**

End of file has been reached before an ENDJOB line has been processed. The ENDJOB line is assumed to have been present immediately before the end of file.

***** ERROR 6 PARAMETER FORMAT IS INVALID**

The format field of a parameter definition line must have the form of the picture clause of a character or display numeric item, except that the word PIC is not used and the maximum character format is X(79). See the Global Development Cobol Language Manual for a full description of valid formats.

*** WARNING 7 PARAMETER PROMPT IS NOT PRESENT**

The current parameter definition line has no parameter prompt string. A null parameter prompt is assumed.

***** ERROR 8 PARAMETER ALREADY DEFINED**

The parameter in the current parameter definition line has already been defined in a previous parameter definition line.

***** ERROR 9 SPECIAL STRING NOT FOLLOWED BY A SPACE**

The special strings <CTRL A>, <CTRL B>, <CTRL C>, <ESC>, <ESCAPE>, <NULL> and <CR> can only appear at the end of a response, and must therefore always be followed by a space or new line.

***** ERROR 10 NO RESPONSE TO COLON IN DIALOGUE LINE**

The current dialogue definition line ends with a colon. Each colon must be followed by a response. This error is often due to a missing <CR> response.

***** ERROR 11 <CR> MUST BE IMMEDIATELY PRECEDED BY A COLON**

When the special string <CR> appears as a response it must form the entire response (i.e. it must be immediately preceded by a colon and followed by a space or new line).

***** ERROR 12 UNDEFINED PARAMETER**

A parameter has been found in the current dialogue definition line, but it has not been defined in a previous parameter definition line in the parameter division.

*** WARNING 13 EXTRA CHARACTERS AT END OF LINE**

Surplus characters have been found at the end of a PARAMETER DIVISION line, DIALOGUE DIVISION line, conditional exit line or ENDJOB line, and have been ignored.

*** WARNING 14 EXTRA LINES AFTER ENDJOB STATEMENT**

Surplus lines have been found following an ENDJOB line and have been ignored.

*** WARNING 15 MISPLACED PARAMETER DIVISION STATEMENT**

A PARAMETER DIVISION line has been found out of sequence and has been ignored.

*** WARNING 16 MISPLACED DIALOGUE DIVISION STATEMENT**

A DIALOGUE DIVISION line has been found out of sequence and has been ignored.

***** ERROR 17 MISPLACED PARAMETER LINE**

A parameter definition line has been found within the dialogue division.

***** ERROR 18 PAUSE LINE NOT TERMINATED BY A MINUS SIGN**

A pause prompt line has been found which is not terminated by a minus sign.

*** WARNING 19 ONLY (SUPPRESS) ALLOWED AFTER DIVISION**

The (SUPPRESS) option only can occur on the DIALOGUE DIVISION line following the word DIVISION. Note that SUPPRESS is enclosed in round brackets.

*** WARNING 20 DIVISION NOT PRESENT**

The word DIVISION is missing or spelt incorrectly in a PARAMETER DIVISION line, but the PARAMETER DIVISION line has been accepted.

*** WARNING 21 DIVISION NOT PRESENT**

The word DIVISION is missing or spelt incorrectly in a DIALOGUE DIVISION line, but the DIALOGUE DIVISION line has been accepted.

***** ERROR 22 LINE STARTING WITH AMPERSAND NOT VALID PARAMETER**

A parameter definition line starting with an ampersand must have one or two digits only immediately following the ampersand and these must be followed by a blank or tab. Moreover the digits must represent a number in the range 1 to 20 inclusive.

***** ERROR 23 PARAMETER FORMAT AND PROMPT NOT PRESENT**

A parameter definition line does not contain a format field or parameter prompt following the parameter number.

***** ERROR 24 INVALID PARAMETER IN DIALOGUE LINE**

A parameter number found as a response in a dialogue definition line is not in the valid range 1 to 20 inclusive.

*** WARNING 25 PAUSE MUST CONTAIN AT LEAST ONE CHARACTER**

There must be at least one character between the leading and trailing minus signs of a pause prompt. A single space character has been assumed.

***** ERROR 26 JOB MUST HAVE SOME DIALOGUE**

No dialogue has been generated for this job file. The dialogue division of every job description must contain at least one dialogue definition line containing a response.

***** ERROR 27 TOO MUCH DIALOGUE**

The dialogue generated from the job description has exceeded the capacity of $JOB's dialogue table. This capacity will depend on the size of the user area in which $JOB is running.

*** WARNING 28 PARAMETERS OUT OF ORDER**

The current parameter definition line has a lower parameter number than the highest parameter number previously defined. It is good practice to supply parameter definition lines in

parameter number sequence, because that is the sequence in which the job file will prompt for them.

*** WARNING 29 CONDITIONAL EXIT LINE OF NO USE**

The current conditional exit line refers to a conditional parameter, and either this parameter or a higher-numbered one has been referred to in a previous conditional exit line. In either case this conditional exit can never cause an exit when the job file is run.

***** ERROR 30 INVALID CONDITIONAL EXIT LINE**

A conditional exit line starting with a question mark must have one or two digits only immediately following the question mark and these must be the only characters on the line. Moreover the digits must represent a number in the range 1 to 20 inclusive.

*** WARNING 31 PARAMETER IS NON-CONDITIONAL**

The parameter referred to in the current conditional exit line has not been defined as conditional in the parameter division. The conditional exit line has been generated, but will have no effect when the job file is run.

***** ERROR 32 MESSAGE NOT TERMINATED BY A PLUS SIGN**

A message line has been found which is not terminated by a plus sign.

*** WARNING 33 MESSAGE MUST CONTAIN AT LEAST ONE CHARACTER**

There must be at least one character between the leading and trailing plus signs of a message. A single space character has been assumed.