COMMERCIAL SYSTEMS

JOBS

USER'S GUIDE

055-042-00

DataGeneral
4400 COMPUTER DRIVE, WESTBORO, MASSACHUSETTS 01580

A Small Business Systems Publication

For the latest enhancements, cautions, documentation changes and other information on this product, please see the release notice supplied with the software.

CS Commercial Systems Documents

Overview and Planning

Concepts and Facilities *055-001*

Gives an overview of the entire CS computer line. The capabilities of the system for business use are described. A separate chapter is devoted to file management. Special attention is paid to CS's unique screen extension.

Planning Guide *055-002*

Administrators and managers will find this guide useful for selecting a CS System. Basic system components are discussed. Options for future upgrading and expansion are presented so that the best possible decision can be made. The book contains complete information on planning and installation. Also included: site preparation checklist, a floor planning kit including punchouts of system components.

Language and System Operation

Interactive COBOL Programmer's Reference *045-011*

In this book, COBOL applications programmers will find all the information needed to write COBOL programs and use the Interactive COBOL screen extension. The book is laid out for quick reference, with an index on the back cover. COBOL verbs are described alphabetically. The use of the compiler is explained, with a list of error messages. Information on the use of the debugger is included, with a list of commands. There are many examples throughout the text, and a chapter containing five sample programs. This is a valuable reference book. Also included: a glossary of Interactive COBOL and standard terms, and a list of COBOL reserved words.

Converting COBOL Programs: CS Interactive COBOL to AOS COBOL *045-009*

This manual contains guidelines for converting existing Interactive COBOL code to equivalent AOS COBOL code. It includes procedures for exporting data and program files from a CS System to an AOS Eclipse System.

Operating CS Systems: A Guide for the System Developer *045-012*

This book documents in detail the software tools that system analysts and system developers will use to operate CS Commercial Systems. File structures and the Interactive COBOL runtime systems are explained. The operating system's command language is defined. Use of utilities, including JOBS batch stream processor, disk initialization and hardware diagnostics is described.

IC/RJE80 Remote Job Entry Communications Utility User's Guide *055-040*

IC/RJE80 implements the nearly-universal inter-computer communications protocol—2780/3780. The manual describes its features and provides detailed operating instructions. It has a tutorial, an alphabetical command listing, a glossary and error messages.

JOBS: Job Organization Batch Stream *055-042*

JOBS is a utility for end-of-day batch processing. JOBS is used to place CLI macros and COBOL programs on end-of-day queue. It then runs the entire queue on command. This manual describes how to use JOBS and illustrates some of the ways it can be useful.

Developer's Tools

ICEDIT: Interactive COBOL Editor *055-004*

This manual describes ICEDIT, a line-oriented source program editor specifically created for the writing of Interactive COBOL programs. Included are details of ICEDIT's interactive operation that makes it particularly easy to learn and use. ICEDIT features an interface with other CS system utilities, including the Interactive COBOL compiler.

CRTEDIT: Display Terminal Text Editor *055-005*

This manual describes CRTEDIT, a powerful string-oriented editor. CRTEDIT maintains a window into the file being edited to assure swift and accurate operation. The editor's flexible command set supports execution of stored editing routines (including conditional branching), use of cursor-positioning keys, cut-and-paste functions, multiple-file handling in a single editing pass, and paragraph-width control for prose composition.

SCREEN: Screen Format Editor *055-006*

This manual describes SCREEN, a special-purpose editor for the design and coding of display screen formats. These formats define the communication between program and operator in Interactive COBOL applications systems. SCREEN users can code new formats and adapt existing formats to different uses.

MASTER Menu and Security System: Guide for the System Developer *055-007*

This technical overview of the MASTER system tells programmers and system developers what they need to know to use MASTER's menu and security features in a business application.

PROXI: COBOL Program Generator *055-038*

This manual describes PROXI, the CS COBOL Program Generator. Included are descriptions of the types of COBOL program generated by PROXI, along with complete instructions for using PROXI. Also included are descriptions of the demonstration programs that are part of the PROXI software package.

Business User Manuals

MASTER Menu and Security System: Security Manager's Guide *055-008*

This manual tells the user with security manager responsibilities how to implement the security features of MASTER in his firm's environment. It is intended to be used in conjunction with the User's Guide.

MASTER Menu and Security System: Supervisor's Supplement *055-009*

This guide is intended for the user with supervisory responsibility in the MASTER system. It is a supplement to the User's Guide.

MASTER Menu and Security System: User's Guide *055-010*

This most basic description of how to use MASTER is intended for anyone who will be working at the terminal using the MASTER system. It is written in non-technical terms, and shows the user how to apply the system's features in day-to-day operations.

TABLE OF CONTENTS

PREFACE

This manual is written for developers engaged in the design of batch job processing operations for an applications system. Chapter 1 gives an introduction to the Job Organization Batch Stream (JOBS) utility and an overview of some of the concepts internal to JOBS. An understanding of them is necessary to use JOBS well. Chapter 2 presents the step-by-step instructions for implementing JOBS.

CHAPTER 1 JOBS CONCEPTS

INTRODUCTION

Job Organization Batch Stream (JOBS) is a batch job processing tool that can automatically run many types of jobs with minimal operator supervision. Since JOBS is designed for use as an end-of-day job processor, the jobs executed by JOBS may consist of programs that require lengthy computations or that generate long printouts. Or a job may require the execution of several COBOL programs in an ordered sequence.

The primary function of JOBS is to build and process "job files." These are the files that contain the instructions for performing specific work routines. In this document a "job file" is defined as a file containing a list of commands to perform a particular function. A "job stream" is the actual list of commands itself, regardless of the fact that those commands are contained in a file. JOBS permits a job stream to be designed which uses both COBOL programs and CLI macros, a facility which gives it quite a bit of power.

User Involvement

Essentially, JOBS requires a user to perform only three operations to execute daily batch processing.

1. Initial setup of a job is required. At this stage, all of the steps that you want to make up a particular job are listed in a job file. You build a library of job files to be used in repetitive end-of-day processing.

2. Each day you select the job files that will be run by JOBS at the end of that day. These go onto the JOBS queue.

3. At the end of each day you order the execution of the JOBS queue. All of the job files that came from the Job File Library and were selected to be run that day are executed.

Before end-of-day processing of the job files in the queue, JOBS generates a setup report for validation purposes. During processing, an operator is normally required only for changing the printer form or for mounting or dismounting, such as loading or unloading a reel of tape. The operator can monitor JOBS and start or restart a job process as necessary. After processing, JOBS prints a report of all jobs executed.

Work Routines

You can use JOBS to fit many job processing requirements. Routines can be as comprehensive and complex as necessary or as straightforward as

printing an end-of-day transaction log. For instance, the following examples illustrate some ways in which JOBS can be applied.

* A job file called Backup.JF could be designed which includes a large number of file sorts and reorganizations. The job stream could then dump the contents of selected files to different positions on a dump tape. With this procedure designed as a job file, the ease of doing a backup is increased and the chance of making errors is reduced.

* A job file named Report.JF could be designed. It can contain many of the steps needed to generate a commonly demanded report. Because a JOBS job file can mix COBOL programs and CLI commands, the report preparation procedure can jump back and forth between the two as required. When the report is prepared, the job file orders the printing of the report.

The following is a sampling of work routines that can be processed by JOBS.

* Run COBOL programs to:

  - Generate reports or a series of related reports.

  - Generate printouts of file maintenance logs that list changes made to files during the day or any other designated period.

  - Generate reports accounting for transactions made during the day or any other designated period.

* Run CLI macros to:

  - Validate (test) data bases.
  - Sort files.
  - Reorganize files.
  - Back up files.
  - Compile COBOL programs.
  - Execute test streams.

All of the above operations may also be mixed in a single job file.

COMPONENTS AND ELEMENTS

JOBS builds and executes job files by using "components" and "elements." JOBS components (Appendix A) are the programs and files supplied in the JOBS software package. The components are COBOL programs, macro files, report files, and operations files. They are used to run JOBS. Internal to JOBS, these components are known as the Catalog Procedures Program (CPP). You can recognize the letters "CPP" attached to various filenames and command lines. Components are used to initiate JOBS, create job files, process job files, and print JOBS reports. The

components are largely invisible when using JOBS, and are executed without need for operator interaction. (Three components that are not invisible and that you will be using often are PREPROC, INITPROC, and STARTCPP.)

JOBS elements (Appendix B) are files, macros, and programs which you can use as the commands in a job file. They are the commands that are valid in a job stream. Elements consist of the following:

* CLI command files--DO macros, macros, indirect filenames.

* .SV files--CLI utility programs.

* COBOL programs.

* .QF files--queue files containing a series of COBOL program names.

* .LJ files--existing macro files to be executed via Local Job Entry.

* .JF files--job files containing elements.

* BRANCH and TAGNAME clauses--special clauses that allow for branching from one element to another within a job stream.

* COMMENT--comment lines for documentation purposes.

DEVELOPMENT AND APPLICATIONS LEVELS

The building and processing of job files occur at two levels of operation--development and applications. At the development level you use JOBS components to build job files of JOBS elements. You are creating job files which will then become jobs in the JOBS Library. At the applications level you use other JOBS components to run the job files which have been built.

Building Job Files At The Development Level

Building a job file is a two-step process that involves:

1. Creating files of valid JOBS elements, and

2. Using these element files to create job files with the JOBS program PREPROC.

See Appendix C for a sample job file.

You can create element files (describing the steps in the job stream) by using an editor. You then call PREPROC to create job files (known by their .JF extension). For instance, you can use the editor ICEDIT to

create a file containing some macro calls to sort and print. You then call PREPROC to transform the file into an executable job file (.JF).

You can also use a free-form facility of PREPROC to directly enter elements into job files created by PREPROC. With this option only PREPROC is used; an editor does not need to be called first. In both operations, PREPROC detects invalid elements and provides for editing. them. After creating a job file, PREPROC enters the job filename (with a .JF extension) in the Job File Library.

JOBS processes elements in the sequential order that they are entered in a job file. However, the special clauses--BRANCH and TAGNAME--provide the capability to alter the sequential order if required by a work routine.

Job filenames can consist of up to ten alphanumeric characters and are entered at the time of job file creation. A .JF extension is automatically appended to the filename at the time of creation. All filenames are listed in the Job File Library, which also includes a description of each job file.

In all cases, PREPROC must first be used to create valid job files before those jobs can be run. JOBS can only process valid job files.

Running Jobs At The Applications Level

Processing begins when the following sequence of operations has been executed:

1. Job files have been selected from the Library and entered into the JOBS queue file;

2. The Interactive COBOL Runtime System has been shutdown, and

3. The command to start JOBS has been entered.

At that time, JOBS generates a printed setup report detailing the work routines to be run, and automatically begins processing the job files in the queue. At the end of processing, JOBS generates an execution log showing which jobs were run (or not run) and any abnormal conditions.

During processing, the screen constantly displays current information on the status of job executions. Messages embedded in job files are also displayed for the operator. If an abnormal or error condition aborts JOBS, a restart program will restart JOBS at any point in the processing. This allows JOBS to continue without immediate correction of the problem by bypassing the problem area.

CHAPTER 2 JOBS OPERATING INSTRUCTIONS

USING PREPROC

PREPROC is an Interactive COBOL program you call from either the MASTER system as a menu choice or from the LOGON "RUN PROGRAM" prompt. Refer to the MASTER System Developer's Guide for information on making a COBOL program one of the menu choices. To run PREPROC from LOGON enter "PREPROC" at the RUN PROGRAM prompt.

PREPROC is the program that creates job files. A job file contains a list of valid JOBS elements, a job stream. You can create the list of elements in two ways.

1. Use an editor to create a file which is a list of JOBS elements. This file is then entered as the INPUT FILENAME to PREPROC. PREPROC makes sure that each of the elements in the file is valid, allows you to edit invalid elements, and creates a job file named Outputfile.JF. If you use an editor to create a file of JOBS elements, there must be only one element per line and it must start at the left margin.

2. Use the free-form facility of PREPROC to enter the job stream. You specify the name of the job file that is created at the OUTPUT FILENAME prompt. The elements you enter are placed directly in the job file. PREPROC puts the new job file in the Job File Library.

The dialogue below shows the questions PREPROC will ask, along with possible answers for each.

Data Prompt	Response
TYPE INPUT FILENAME:	Enter a valid element filename or press <ESC> to end PREPROC. Or press <CR> to call the free-form format.
DO YOU WANT TO TYPE IN FREE-FORMAT? (Y OR N)	Press <Y> to go to the OUTPUT FILENAME prompt. Press <N> to return to the INPUT FILENAME prompt.

Data Prompt	Response
TYPE OUTPUT (.JF) FILENAME:	Enter any name for the job file, up to 10 alphanumeric characters. The .JF extension is appended to the filename automatically.
_FILE ALREADY EXISTS PRESS FN1 TO OVERWRITE, FN8 TO APPEND, ESC TO RENAME.	Press FN1 to overwrite job file Outputfilename.JF. APPEND or ESC. Press FN8 to append new elements to Outputfilename.JF. Press <ESC> to rename job file Outputfilename.JF at the OUTPUT FILENAME prompt.
TYPE A PROCEDURE ELEMENT LINE AND PRESS CR. (ESC TO END)	For free-format, type in a valid element and press <CR> to end each element line. Press <ESC> to end free-form input to the job file and proceed to the TITLE: prompt.
INVALID PROCEDURE ELEMENT PLEASE RETYPE AND PRESS CR.	Retype the correct element and press <CR>.
DELETE JOB FILE (Y OR N)	Press <ESC> during editing of an input element file to generate the DELETE JOB prompt. Press <Y> to delete the job file and return to the main menu. Press <N> to return to the main menu.

Data Prompt	Response
TYPE A COMMENT FOR THIS CPP ELEMENT, OR PRESS CR.	Type in a comment to describe the element and press <CR>. This comment will be listed on all reports. Press <CR> to bypass comment entry.
TITLE:	You must type in a title to describe the job file. Type title--up to 32 characters --and press <CR>.

NESTING JOB FILES

In building job files, you can nest job files within other job files. For example, a job file containing elements for generating and printing a standard report could be nested in another job file that requires such a report. Nesting eliminates entering the same elements repetitively in the other job file.

To nest a job file within another file, enter the Filename.JF in the specified job file as you would for any element. Job files can be nested up to sixteen levels deep, beginning with level "A" and proceeding through level "P".

An A-level job file is defined as the first job file in a hierarchy of nested job files. In other words, an A-level job file can contain other job files, but is not itself contained in a job file. JOBS can process up to 998 level A job files.

JOB STEPS AND NUMBERING SYSTEM

Within a job file, JOBS executes elements in a sequential progression of job steps. In other words, the order in which you specify the execution of elements determines the order of execution of job steps.

As part of the processing routines, JOBS uses a seven-character, alphanumeric system to number job files and job steps (000.000A-P). JOBS creates the numbering system and uses it for its own control. You do not enter these numbers when job files are created.

The first three digits are used to identify each level A job file, from 1 through 998. For instance, the first level A job file is numbered 1.000A, the second 2.000A, up to 998.000A. The number 999 is reserved to indicate the end of processing.

The second three digits are used to number each job step in a level A job file, from 1 through 998. Job steps within each level A job file are numbered sequentially, .001, .002, up to .998. The alphanumeric number .999A is used to indicate the end of a level A job file.

Job steps that are also job files include an alphabetic character denoting the level of the job file. For instance, job step 002.089C indicates that step 89 is a level C job file nested in a level B job file that is nested in the second level A job file.

JOBS can process up to 998 level A job files. Each level A job file can contain up to a combined total of 998 JOBS elements and nested job files, for a total of 998 job steps (Figure 2.1).

```
   1.000A       Job File A
   1.001        Element in Job File A
   1.002        Element in Job File A
   1.003B       Element in Job File A--Job File B
   1.004C       Element in Job File B--Job File C
   1.005        Element in Job File C
   1.006D       Element in Job File C--Job File D
   1.007        Element in Job File D
   1.008D       Element in Job File C--Job File D end
   1.009C       Element in Job File B--Job File C end
   1.010B       Element in Job File A--Job File B end
   1.011        Element in Job File A
   1.012        Element in Job File A
   1.999A       Element in Job File A--Job File A end

   2.000A       Job File A
   2.001        Element in Job File A
   2.999A       Element in Job File A--Job File A end

 999.999        Finale
```

Figure 2.1 Job Step Numbering With Nesting

CONDITIONAL BRANCHING

Branching is the process of moving to job steps previous to the current job step or ahead of the next job step. Branching may be required when a work routine is dependent on another work routine being executed first.

The procedure for conditional branching in JOBS involves three steps:

1. Placing labels in the job stream which will be used as reference points when branching occurs;

2. Entering conditional branching statements in the job stream, and

3. Changing the value of a variable that is used in the conditional test.

You enter labels in the job stream with the #TAGNAME element. The label is simply a reference point that the conditional branching element will use later as a location to branch to.

You then enter conditional branching commands with the CONDITIONnn#TAGNAME element. This statement says, in effect, "when the variable has the proper relationship to the fixed value, go to the label."

Finally, you change the value of the variable by operating on a file called CPPRETURN.

The following three steps contain specific instructions for conditional branching. In a JOBS job file:

1. Enter the reference point for branching with the #TAGNAME clause. Enter #TAGNAME into a job file immediately preceding the step that you want the branch to go to. When a branch to that label is taken, the steps following that label will be executed. A label can contain up to 24 characters.

   Example: #BRANCHLOC3

2. Enter the conditional branching statement with the BRANCH clause. The format is as follows:

   ```
   [ <  ] yy#TAGNAME          yy = any value to be
   [ >  ]                          compared with
   [ =  ]                          the variable
   [ <= ]                          (discussed in step 3)
   [ >= ]
   [ <> ]
   ```

   Example: = 12#BRANCHLOC3

   The example is interpreted to mean: If the variable is equal to 12, go to the label BRANCHLOC3. Note: A single conditional symbol must be followed by a space. For example, do not enter "<12#BRANCH2"; rather, enter "< 12#BRANCH2".

3. Manipulate the variable that is used in the conditional branching test. Whenever a conditional branch statement is encountered, the two-digit value included in it is compared with the first two characters in a file called CPPRETURN. This means that in the example in step 2, 12 is compared with the first two characters in CPPRETURN. If the two values are equal, the branch to BRANCHLOC3 is taken.

   You can change the value in CPPRETURN in a number of ways. There is a JOBS element, BRANCHSET, which puts a two-digit value into CPPRETURN. The syntax for using this is: DO BRANCHSET "xx", quotation marks included. The value "xx" can represent any two alphanumeric characters. You can enter a comment string, but BRANCHSET will use only the first two characters to set a value in CPPRETURN.

   Perhaps the most useful way of changing CPPRETURN is within a COBOL program. Various conditions in a program can set CPPRETURN. The branching that is taken by JOBS can then be based on the value that was set in the COBOL program. If you create CPPRETURN in a COBOL program, it must be created as a line file. Assigning the file to PRINTER or DISPLAY accomplishes this task.

   CLI commands that put characters into a file can be used to alter CPPRETURN as well. As with CPPRETURN in a COBOL program, the file produced by CLI commands must end with a line terminator (<CR> or<NL>).

QUEUING JOB FILES

From LOGON

After you build job files they must be queued for processing each time you want them to run. This requires entering the job filenames into the job queue file with the COBOL program INITPROC (Figure 2.2).

From LOGON, INITPROC involves two steps. After the program is invoked from the RUN PROGRAM prompt, you enter the name of the job file. You are then asked to confirm its entry to the queue. That job file and all of the others that are queued will run the next time the JOBS queue is executed (see "Executing JOBS" section of this Chapter). Each time the JOBS queue is run, the queue is emptied. Another list of jobs must be entered for the next running.

When INITPROC is run from LOGON, the following dialogue occurs.

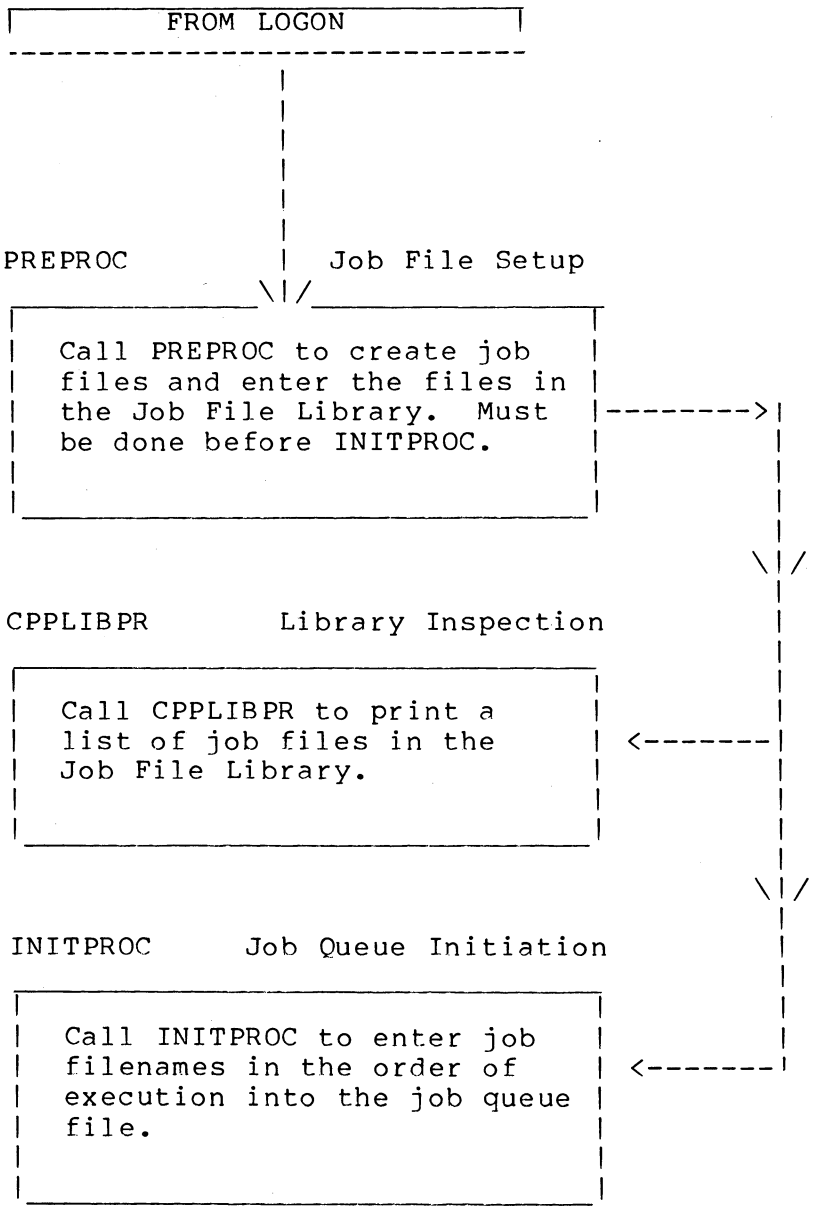

Figure 2.2 Preparing and Queuing Jobs from LOGON

INITPROC Dialogue

Data Prompt	Response
TYPE CPP PROCEDURE NAME:	Type in job filename and press <CR>. Press <ESC> to return to LOGON.
PRESS ESC TO BY-PASS INITIATION OR PRESS CR TO CONTINUE.	Press <ESC> to withhold the job filename from the queue file. Press <CR> to enter the job filename into the queue file. If <CR> is pressed, the TYPE CPP prompt is returned for entry of the next job filename.

From MASTER

Queuing jobs from MASTER is essentially the same as from LOGON, except that INITPROC is run as one of the menu choices. In each of the methods described below, the user will see INITPROC placing the job on the queue, but will have no interaction with INITPROC; he or she will not be asked to confirm the queue.

A job file can be queued in three ways from MASTER.

1. Enter "!job-filename.JF" at the PROGRAM CALLED prompt. The "!" invokes INITPROC/D which, in turn, places the job on the queue.

2. Enter "!!program-name" at the PROGRAM CALLED prompt. This is a way to place a COBOL program that is not a job file directly on the queue. JOBS does this by reserving a job file named TCUQ033.JF. The two exclamation marks cause the COBOL program to be placed in a file that is executed by this job file.

3. If the name of the above program and its switches is longer than 10 characters, you can use the format "!!#line-file". Store the program name and its switches in the line file. This is a way to indirectly reference a long program name.

A note of caution about the latter two methods is needed, however. All of the COBOL programs that are queued in this way (directly) will be run together. Since all of the programs will be on the queue in one file, they will be run immediately following one another. TCUQ033.JF will be on the queue in the position determined by the first direct program entry.

This situation may cause particular problems if one of the programs queued in this way changes the value in CPPRETURN. JOBS treats the set of programs queued directly as one unit. CPPRETURN will not be available for conditional branching until all of the programs queued directly are finished.

EXECUTING JOBS

After job filenames have been entered into the queue file, the queue can be executed (Figure 2.3); however, the COBOL Runtime System must be shut down first. JOBS is then run either directly from the CLI R prompt or directly following MASTER.

Each time the JOBS queue is executed, JOBS uses the queue file to create a control file. The control file oversees the processing of the elements in the queue. Each time JOBS completes processing, a backup control file that can be rerun is generated for historical purposes. The original control file is deleted and the backup file named with the current day's date.

JOBS will process the job files without need of direct operator involvement unless required by a particular job. For example, an operator may be needed to load a tape when JOBS is doing backups or to restart JOBS when an abnormal condition exists.

Starting JOBS From LOGON

At the LOGON menu, enter <S> to shutdown the COBOL Runtime System and return the CLI R prompt. You can then call JOBS by typing at the R prompt:

```
DO STARTCPP xCI
```

x = EMM, MM, BS, EDS, DS

(xCI is the name of your COBOL interpreter.)

JOBS processing begins, a job setup report is generated, and the elements in the job streams are executed.

Starting JOBS From MASTER

Use the MASTER system startup macro that includes JOBS execution. Type at the CLI R prompt:

```
DO MASTERCPP xCI,N
```

x = EMM, MM, BS, EDS, DS

N = number of terminals

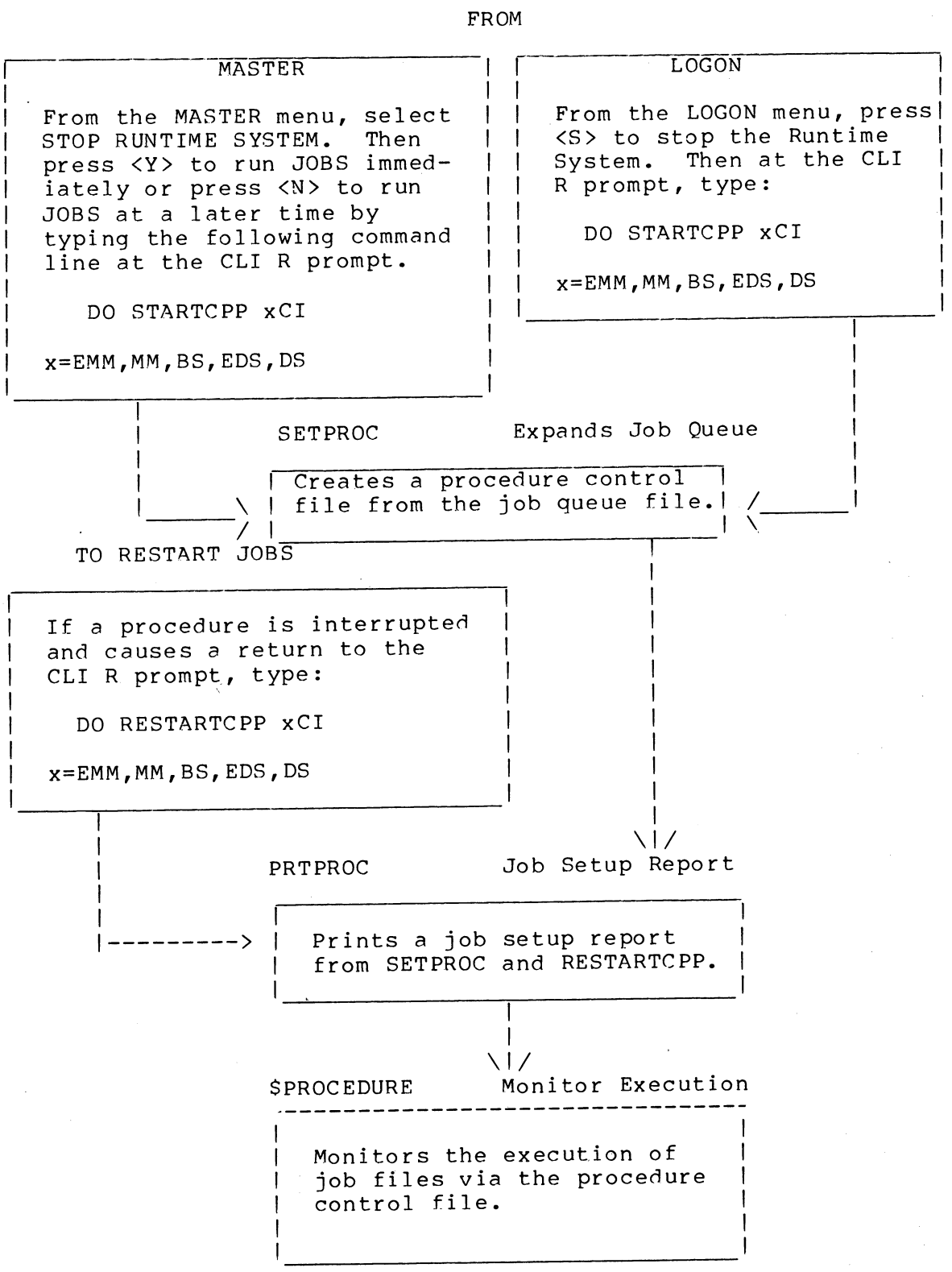

Figure 2.3 Executing Jobs From The Job Queue File

This DO macro runs MASTER in the normal manner, but invokes JOBS when MASTER is shut down. At the completion of using MASTER you call JOBS by selecting SHUT DOWN SYSTEM from the MASTER menu. The following data prompt is then displayed:

Data Prompt	Response
FINAL STOP (Y OR N)	Press <Y> to run JOBS immediately. If a job queue file has been created, JOBS generates a job setup report and begins processing. If no job queue file exists, MASTER returns to the CLI mode. Press <N> to not run JOBS or to run JOBS at a later time. MASTER returns to the CLI mode.

JOBS REPORTS

JOBS generates the following reports:

* CPP - SETUP - REPORT	Generated before processing begins. Lists the jobs to be executed.
* CPP BRANCHING AUDIT REPORT	Lists BRANCH clauses, TAGNAME clauses, and TAGNAME references encountered during creation of the procedure control file in program SETPROC. This report is appended to the setup report.
* CPP EXECUTION LOG	Generated after processing and after a restart. Lists all jobs executed.
* CPP - RESTART - REPORT	Generated at restart. Lists the jobs to be executed.

See Appendix C for sample JOBS reports.

RESTARTING JOBS AT THE CLI

Use the restart macro to restart JOBS whenever processing has been interrupted because of job procedure failures or operator intervention. At the CLI R prompt, type:

DO RESTARTCPP xCI

x = EMM, MM, BS, EDS, DS

(xCI is the name of your COBOL interpreter.)

JOBS then displays a restart menu (Figure 2.4) with several options.

```
---------------------------------------------------------------------------

$PROCEDURE-00          CATALOG PROCEDURES MONITOR

STEP NUMBER:           TYPE:              STARTED:  YY-MM-DD  HH:MM

(Current step...)



RESTART OPTIONS ARE:

        1.  RESTART AT CURRENT STEP (SHOWN ABOVE).
        2.  RESTART AT THE NEXT STEP.
        3.  RESTART AT ANOTHER SPECIFIED STEP.
        4.  STOP FOR REPAIRS OR FURTHER ANALYSIS.

            TYPE THE NUMBER OF YOUR RESTART OPTION CHOICE -

---------------------------------------------------------------------------
```

Figure 2.4 Restart Menu

Note: If one of the elements in a job file is a .QF file, you cannot restart in the middle of it. .QF files contain a list of COBOL programs and are treated as one unit by JOBS. You cannot get inside a .QF file and start at the desired program; you can only restart at the beginning of the .QF file or skip it entirely.

PRINTING A LISTING OF JOB FILES

The COBOL program CPPLIBPR prints a listing and description of all job files in the Job File Library. You can call CPPLIBPR from either the MASTER system or LOGON.

RERUNNING A COMPLETED JOBS QUEUE

There are two ways that you can rerun the entire JOBS queue. You can either recreate the queue and have JOBS execute that queue again or you can recreate the control file that JOBS produces from the queue and run that control file.

To recreate the queue use the command: DO RESETCPP xCI,yymmdd. xCI is the name of your COBOL interpreter. yymmdd is the date of the queue that you want run. This macro uses a backup queue file that is labeled with the day it was created. The queue is rebuilt and JOBS is run exactly as it was on that day.

To recreate the control file (built from the JOBS queue) use the command: DO RERUNCPP xCI,yymmdd. This macro uses a backup control file. If you rerun JOBS in this way, you will be shown the restart menu. You may then select the job step that you want to start at.

Note: JOBS keeps all of its backup control files so that they can be used for a rerun. Over a period of weeks and months the number of backup files will build up. If you want to clean up the accumulated backup files, look for the following names on your storage media: CPPQyymmdd, CPP$yymmdd, CPPTyymmdd.

APPENDIX A JOBS COMPONENTS

This Appendix of JOBS components is included for those who would like to know some of the internals of JOBS operation. Most of the programs and files described below are invisible to the user. It is not necessary to understand all of the components that make up JOBS in order to use it well.

Table A-1 COBOL Programs

Name	Description
PREPROC	An interactive program to build job files (.JF files). Can be called either from the MASTER system by menu choice or from LOGON by typing "PREPROC" at the RUN PROGRAM prompt. Transforms a line file containing valid JOBS elements into a job file. Or creates a job file by accepting the direct entry of elements in free-format. Rejects invalid elements and provides for editing. Creates and maintains a library file of job filenames. Refer to Table A-4, operations file PROCLIBR.<NX XD>. Refer to Table A-4, operations file filename.JF.
INITPROC	An interactive program used to queue job filenames into the JOBS CPP queue file. Can be called from either the MASTER system by menu choice or from LOGON by typing "INITPROC" at the RUN PROGRAM prompt. Creates a CPP queue file, PROCQUEUE.JF. Validates that the job filename exists in the CPP library file, and enters the job filename into the CPP queue file. Refer to Table A-4, operations file PROCQUEUE.JF.
INITPROC/D	Automatically queues job filenames that exist as MASTER system menu choices into the CPP queue file. Examines the MASTER control file for selected choices, creates a CPP queue file, validates the job filenames, and enters the job filenames into the CPP queue file.

Table A-1 COBOL Programs (Continued)

Name	Description
	Runs only by call from $PROCEDURE as the first step in processing.
SETPROC	Expands all job files into elemental job steps and builds a JOBS CPP control file. Creates backup queue file CPPQyymmdd where yymmdd is the current day's date. Refer to Table A-4, operations file CPPQyymmdd.
$PROCEDURE	Linked to LOGON and substitutes for the LOGON program during JOBS execution. Monitors the execution of jobs via the operations file PROCEDURE.<NX XD>.
	Prints the JOBS setup and restart reports.
PRTPROC	Creates the operations file CPPLOG that contains a copy of the setup and restart reports. Refer to Table A-4, operations file CPPLOG.
CPPLIBPR	Prints a listing of all job files in JOBS CPP library file PROCLIBR.<NX XD>. Listing includes the following: Filename Description Date of creation Date last accessed Refer to Table A-4, operations file PROCLIBR.<NX XD>.
TCUQ	Used to handle programs queued by !!progname. Reads the program and terminal record from TCUQ033.FQ. Is entered in TCUQ033.QF for each program that is placed in TCUQ033.FQ by a !!progname call.

Table A-2 DO Macro Files

Name	Description
MASTERCPP	MASTER system startup macro that includes the CPP. Execute by typing at CLI R prompt: DO MASTERCPP xCI,N xCI is the name of your COBOL interpreter. N = number of terminals Calls the MASTER system for runtime operation. At shutdown of runtime system, calls JOBS to process any job files that have been placed in the queue file via INITPROC.
STARTCPP	JOBS startup macro. Execute by typing at CLI R prompt: DO STARTCPP xCI xCI is the name of your COBOL interpreter. Calls JOBS to process job files.
RESTARTCPP	JOBS restart macro used when an interruption causes a return to the CLI. Execute by typing at CLI R prompt: DO RESTARTCPP xCI xCI is the name of your COBOL interpreter. Displays a restart menu with the following choices. 1. Restart at the current step. 2. Restart at the next step. 3. Restart at another specified step. 4. Stop for repairs or further analysis. To restart processing at or prior to a "failed" step, first you may need to correct the condition that caused the failure.

Table A-2 DO Macro Files (Continued)

Name	Description
$PROCSETUP	Included in MASTERCPP, STARTCPP, and RESTARTCPP Creates the CLI macro files required to establish and control processing.
CPPWRAPUP	Executed from the $PROCEDURE program as the last step in processing. Executed as DO CPPWRAPUP yymmdd. Creates a line file version, CPP$yymmdd, of the CPP control file for backup purposes where yymmdd is the current date. Deletes the CPP control file.
RERUNCPP	Used to reorganize a CPP$yymmdd file into a CPP control file and to restart JOBS. Execute by typing at CLI R prompt: DO RERUNCPP xCI,yymmdd xCI is the name of your COBOL interpreter.
RESETCPP	Used to convert a CPPQyymmdd file into a JOBS queue file and to start JOBS. Execute by typing at CLI R prompt: DO RESETCPP xCI,yymmdd xCI is the name of your COBOL interpreter.

Table A-3 CLI Macro Files

Name	Description
$PROCEDURE	Contains initial CLI commands. Contents transferred to macro file $PROCEDURE.MC when the DO macro $PROCSETUP is executed.
$RUNPROC	Initializes macro files used in processing. Contents transferred to macro file $RUNPROC.MC when the DO macro $PROCSETUP is executed.
$DOPROC	Initializes macro files used in processing. Contents transferred to macro file $DOPROC.MC when the DO macro $PROCSETUP is executed.
$RUNPROC2	Contains @xCI/I/B@ after execution of the DO macro $PROCSETUP. xCI is the appropriate COBOL interpreter name.
$PROCEDURE.MC	Contents initially the same as those in $PROCEDURE. Contents entered into macro file $PROCEDURE.MC when the DO macro $PROCSETUP is executed.
$RUNPROC.MC	Contents initially the same as those in $RUNPROC. Contents transferred to macro file $RUNPROC.MC when the DO macro $PROCSETUP is executed.
$DOPROC.MC	Contents initially the same as those in $RUNPROC. Contents transferred to macro file $DOPROC.MC when the DO macro $PROCSETUP is executed.

Table A-4 Operations Files

Name	Description
filename.JF	Created by the PREPROC program from an input file containing valid elements. Each .JF filename is listed in the CPP library file PROCLIB.<NX XD>.
PROCQUEUE.JF	Initially created by and appended to the INITPROC program. Lists job files for processing. This is the daily queue file.
PROCQUEUE.TF	Contains user and terminal records for each job file queued under MASTER. Used to keep track of which company is running a particular program under JOBS.
PROCEDURE.<NX XD>	Indexed sequential file (ISAM) created by the SETPROC program as it reads in the listing of job files in PROCQUEUE.JF. Used as the control file by $PROCEDURE during the execution of the job stream.
$PROCEDURE.SW	Created by DO macro RESTART to indicate the restarting of JOBS.
CPPLOG	Contains a copy of the setup report and restart report, if any. Identical to listing produced by the system printer. File is overwritten each time the SETPROC program runs.
TCUQ033.JF	Entered on the JOBS queue the first time a program is queued directly from MASTER using a !!progname call. Contains the element TCUQ033.QF.
TCUQ033.QF	Contains the program TCUQ for each program entered to TCUQ033.FQ by a !!progname call.

Table A-4 Operations Files (Continued)

Name	Description
TCUQ033.FQ	TCU control file. Contains program name and terminal record for each !!program queued.
PROCLIBR.<NX XD>	Library file maintained by PREPROC and INITPROC. PREPROC adds new job filenames. INITPROC deletes the name of .JF files that have been deleted from the system's master directory. Contains the following information for each job filename: 1. Date created. 2. Date and time last used. 3. 32-character description line. 4. Type of last access: origin, modify, init, start, end.
CPPRETURN	The conditional value set by the DO macro BRANCHSET "xx" is entered into this file and used in comparison with the fixed value set by the BRANCH clause. Refer to Appendix B, DO BRANCHSET and BRANCH clauses.
$PROGPROC.<NX XD>	Temporary file created from a .QF file of COBOL programs. Programs listed in the queue file are executed in the order of their listing. A queue execution report is printed on the system printer after the last program has been run. File deleted after execution of programs.
$PROCLOG	Created by program $PROCEDURE. Contains execution log information accumulated during processing. Application programs and CLI procedures may append related job execution information to this file.

Table A-4 Operations Files (Continued)

Name	Description
	Created by the SETPROC program. Backup file for the CPP queue file which is deleted after the CPP control file has been built.
CPPQyymmdd	Used in the DO macro RESETCPP. yymmdd is the current date.
	Created by the DO macro CPPWRAPUP. Backup file for the CPP control file, which is deleted at the end of normal processing.
CPP$yymmdd	Used in the DO macro RERUNCPP. yymmdd is the current date.
CPPTyymmdd	Backup to PROCQUEUE.TF. Used if JOBS is run again to keep track of which company queued a particular program from MASTER.

APPENDIX B JOBS ELEMENTS

Table B-1 JOBS Elements

Element	Element Format	Description
Macro File	Filename.MC	File must contain CLI commands in order of execution.
Indirect	@Filename@	File must contain CLI commands in order of execution.
DO Macro	DO Filename [arg1,...argn]	File must contain CLI commands in order of execution and parameters.
Job File	Filename.JF	File must contain valid elements (which may include other .JF files).
Local Job File	Filename.LJ	File must correspond to an existing .MC file to be executed via LJE. As this is an indirect reference to a .MC file, Filename.LJ itself is not executed and must not currently exist.
COBOL Program	PROGNAME [/A/B/C.../Z]	Existing COBOL program--switches optional.
Queue File	Filename.QF	File must contain a series of COBOL program names, one name per line, with switches optional.

Table B-1 JOBS Elements (Continued)

Element	Element Format	Description
Save File	Filename.SV	File must contain a valid CLI utility; may include switches and arguments.
TAGNAME Clause	#TAGNAME	Non-executable procedure element used as a point of reference by the BRANCH clause. TAGNAME can be any string up to 24 keyboard characters.
BRANCH Clause	[<=] [>=] [=] yy#TAGNAME [<] [>] [<>]	Sets a fixed value for comparison with the conditional value set by the DO macro BRANCHSET. When the condition is true, CPP branches to the next element that follows the corresponding #TAGNAME clause. yy can be any two alphanumeric values.
DO Macro BRANCHSET	DO BRANCHSET "xx" or DO BRANCHSET "xx" COMMENT	Sets a conditional value for comparison with the fixed value set by the BRANCH clause. When the comparison is true, CPP branches to the procedure following the #TAGNAME clause. "xx" represents any two alphanumeric characters. You can enter a longer string, but a BRANCH clause will use only the first two characters.

Table B-1 JOBS Elements (Continued)

Element	Element Format	Description
Comment	/any comment	Comment line which may be inserted for purposes of documentation.
		If comment lines include any of the following, they are automatically edited and changed.

Original Value	Changed To
(	'
)	'
<	'
>	'
[	'
]	'
@	#

APPENDIX C SAMPLE JOBS REPORTS

Sample JOBS Setup Report

```
10/14/79 16:48    CATALOGUED PROCEDURE -SETUP- REPORT    REV 1.00    PAGE 1

JOB.STEP  TYPE          ELEMENT DESCRIPTION                STARTED  END/XREF
--------  ----------    ---------------------------        -------  --------

  1.000A  JOB FILE      BRNCHTEST.

  1.001B  JOB FILE      PROGTEST.
  1.002   QUEUE FILE    PROGQUEUE.QF
  1.003   DO MACRO      DO PRINTEM "MSSLOG"
  1.004   LOCAL JOB     B.LJ
  1.005   DO MACRO      DO BRANCHSET "00"
  1.006B  JOB FILE      PROGTEST.  END                                  1.001

  1.007   BRANCH      IF VALUE =  00 GO TO TAGNAME1                     1.012
                              /TEST BRANCH TO TAGNAME1

  1.008B  JOB FILE      AB0002.
  1.009   DO MACRO      DO BRANCHSET "00"
  1.010   LOCAL JOB     B.LJ
  1.011B  JOB FILE      AB0002.    END                                  1.008

  1.012   TAGNAME ::     TAGNAME1
  1.013   BRANCH      IF VALUE =  00 GO TO TAGNAME2                     1.036
                              /TEST BRANCH TO TAGNAME2
  1.014   TAGNAME ::     TAGNAME3

  1.015B  JOB FILE      DO5.

  1.016C  JOB FILE      DO6.

  1.017D  JOB FILE      DO7.

  1.018E  JOB FILE      DO8.

  1.019F  JOB FILE      DO9.

  1.020G  JOB FILE      DO10.

  1.021H  JOB FILE      DO11.

  1.022I  JOB FILE      DO12.
```

Sample JOBS Setup Report (Continued)

```
  1.023J  JOB FILE     DO13.
  1.024   DO MACRO     DO BRANCHSET "01"
  1.025J  JOB FILE     DO13.       END                                  1.023

  1.026I  JOB FILE     DO12.       END                                  1.022

  1.027H  JOB FILE     DO11.       END                                  1.021

  1.028G  JOB FILE     DO10.       END                                  1.020


  1.029F  JOB FILE     DO9.        END                                  1.019

  1.030E  JOB FILE     DO8.        END                                  1.018

  1.031D  JOB FILE     DO7.        END                                  1.017

  1.032C  JOB FILE     DO6.        END                                  1.016

  1.033B  JOB FILE     DO5.        END                                  1.015

  1.034   CLI MACRO    C.MC
                                   /TEST CLI MACRO FILE
  1.035   BRANCH     IF VALUE >  00 GO TO TAGNAME4                       1.038
                                   /TEST BRANCHING TO TAGNAME4
  1.036   TAGNAME ::    TAGNAME2
  1.037   BRANCH     IF VALUE =  00 GO TO TAGNAME3                       1.014
                                   /TEST BRANCHING TO TAGNAME3
  1.038   TAGNAME ::    TAGNAME4
  1.999A  JOB FILE     BRNCHTEST.  END                                  1.000


999.999                PRINT $PROCLOG                                   16:48
                                   /PRINT PROCEDURES LOG
```

Sample JOBS Execution Report

```
10-14-79  16:48:25   CATALOGUED PROCEDURES EXECUTION LOG      REV 1.00
                                                              PAGE:  1

JOB.STEP  TYPE       ELEMENT DESCRIPTION                     DATE      TIME

--------  ---------  ------------------------------          --------  --------

  1.000A  JOB FILE   BRNCHTEST.                              10-14-79  16:48:27
  1.001B  JOB FILE   PROGTEST.                               10-14-79  16:48:28
  1.002   QUEUE FILE PROGQUEUE.QF                           10-14-79  16:48:29
          0001       BLAHHH                                  10-14-79  16:48:32
PROGRAM NOT FOUND:
                                              ENDED:        10-14-79  16:48:35
          0002       MSSLOG/D                                10-14-79  16:48:36
                                              ENDED:        10-14-79  16:49:10

  1.003   DO MACRO   DO PRINTEM "MSSLOG"                     10-14-79  16:49:13
                                              ENDED:        10-14-79  16:49:34
  1.004   LOCAL JOB  B.LJ                                    10-14-79  16:49:34
                                              ENDED:        10-14-79  16:49:44
  1.005   DO MACRO   DO BRANCHSET "00"                       10-14-79  16:49:44
                                              ENDED:        10-14-79  16:50:04
  1.006B  JOB FILE   PROGTEST.  END   STEP   1.001           10-14-79  16:50:04

  1.007   BRANCH  IF VALUE = 00 GO TO  TAGNAME1               10-14-79  16:50:05
          COMMENT            /TEST BRANCH TO TAGNAME1
          VALUE IS   00
          BRANCH TAKEN.

  1.012   TAGNAME  :: TAGNAME1                               10-14-79  16:50:06
  1.013   BRANCH  IF VALUE = 00 GO TO  TAGNAME2              10-14-79  16:50:07
          COMMENT            /TEST BRANCH TO TAGNAME2
          VALUE IS   00
          BRANCH TAKEN.

  1.036   TAGNAME  :: TAGNAME2                               10-14-79  16:50:08
  1.037   BRANCH  IF VALUE = 00 GO TO  TAGNAME3              10-14-79  16:50:08
          COMMENT            /TEST BRANCHING TO TAGNAME3
          VALUE IS   00
          BRANCH TAKEN.

  1.014   TAGNAME  :: TAGNAME3                               10-14-79  16:50:10
  1.015B  JOB FILE   DO5.                                    10-14-79  16:50:10
  1.016C  JOB FILE   DO6.                                    10-14-79  16:50:12
```

Sample JOBS Execution Report (Continued)

```
  1.017D  JOB FILE   DO7.                                    10-14-79  16:50:13
  1.018E  JOB FILE   DO8.                                    10-14-79  16:50:14
  1.019F  JOB FILE   DO9.                                    10-14-79  16:50:15
  1.020G  JOB FILE   DO10.                                   10-14-79  16:50:16
  1.021H  JOB FILE   DO11.                                   10-14-79  16:50:17
  1.022I  JOB FILE   DO12.                                   10-14-79  16:50:18
  1.023J  JOB FILE   DO13.                                   10-14-79  16:50:19
  1.024   DO MACRO   DO BRANCHSET "01"                       10-14-79  16:50:20


                                          ENDED:             10-14-79  16:50:41
  1.025J  JOB FILE   DO13.      END   STEP   1.023           10-14-79  16:50:41


  1.026I  JOB FILE   DO12.      END   STEP   1.022           10-14-79  16:50:42

  1.027H  JOB FILE   DO11.      END   STEP   1.021           10-14-79  16:50:43

  1.028G  JOB FILE   DO10.      END   STEP   1.020           10-14-79  16:50:44

  1.029F  JOB FILE   DO9.       END   STEP   1.019           10-14-79  16:50:46

  1.030E  JOB FILE   DO8.       END   STEP   1.018           10-14-79  16:50:47

  1.031D  JOB FILE   DO7.       END   STEP   1.017           10-14-79  16:50:48
  1.032C  JOB FILE   DO6.       END   STEP   1.016           10-14-79  16:50:49

  1.033B  JOB FILE   DO5.       END   STEP   1.015           10-14-79  16:50:50

  1.034   CLI MACRO  C.MC                                    10-14-79  16:50:51
          COMMENT            /TEST CLI MACRO FILE
                                          ENDED:             10-14-79  16:51:08
  1.035   BRANCH  IF VALUE > 00 GO TO  TAGNAME4              10-14-79  16:51:08
          COMMENT            /TEST BRANCHING TO TAGNAME4
          VALUE IS   01
          BRANCH TAKEN.

  1.038   TAGNAME  :: TAGNAME4                               10-14-79  16:51:09
  1.999A  JOB FILE   BRNCHTEST. END   STEP   1.000           10-14-79  16:51:09

999.999   FINALE     PRINT $PROCLOG                          10-14-79  16:51:10
          COMMENT            /PRINT PROCEDURES LOG
                                          ENDED:             10-14-79  16:51:32
```

APPENDIX D JOBS ERROR MESSAGES

Table D-1 JOBS Error Messages

Error	Reason/Action
PROCEDURE FILE READ ERROR	Fatal condition.
INITIAL RECORD IN PROCEDURE FILE NOT FOUND	Fatal condition.
THERE ARE NO CATALOGUED PROCEDURES TO BE RUN	No PROCQUEUE.JF.
REWRITE ERROR ON	Fatal condition PROCEDURE FILE.
PROGRAM NOT FOUND	JOBS continues with next program.
INVALID PROCEDURE STEP	JOBS continues with next step.
MACRO PROCEDURE ERROR IN OUTPUT TO filename xx	JOBS continues with next step. xx is an error code.
SYSTEM STOP ERROR	Fatal condition.
LJE INITIATION PROGRAM NOT FOUND	JOBS continues with next step.
SYSTEM END ERROR	Fatal condition.
QUEUE FILE NOT FOUND xx	JOBS continues with next step. xx is an error code.

reader comment form

JOBS User's Guide

055-042-00

Your comments will help us improve the quality of this publication. If your answer to a question is "NO" or requires qualification, please explain.

Name _____ Title _____
Firm _____ State _____
Address _____ Zip _____
City _____ Date _____

HOW DID YOU USE THIS PUBLICATION?

- ☐ As an introduction to the subject
- ☐ For information about operating procedures
- ☐ To instruct in a class
- ☐ As a student in a class
- ☐ As a reference manual
- ☐ Other *(please explain):*

DID YOU FIND THE MATERIAL:

	YES	NO		YES	NO
• Useful	☐	☐	• Well illustrated	☐	☐
• Complete	☐	☐	• Well indexed	☐	☐
• Accurate	☐	☐	• Easy to read	☐	☐
• Well organized	☐	☐	• Easy to understand	☐	☐
• Well written	☐	☐			

We would appreciate any other comments. Please refer to page numbers if appropriate. Your comments will be carefully reviewed by the writers of this document.

COMMENTS:

CUT ALONG DOTTED LINE

First fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 WESTBORO, MASS 01580

POSTAGE WILL BE PAID BY ADDRESSEE:

DataGeneral

ATTN: Small Business Systems Documentation
MS F213
Westboro, Ma. 01580
USA

Second fold

PLEASE TAPE CLOSED (NO STAPLES)