JOB CONTROL PROGRAM

USER'S MANUAL

P.O. Box 49302
Austin, Texas 78765

WARRANTY

The information in this manual and the software supplied has been carefully checked and is believed to be reliable and functional; however, such material is sold "as is" and PETER MURRAY AND FRANK HOGG DENTAL LAB GRANT NO WARRANTY OF ANY KIND AS TO SOFTWARE OR THE MANUAL, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS. This is in lieu of all liability or obligations of Peter Murray and Frank Hogg Dental Lab for all damages including, but not limited to, consequential damages.

IN NO EVENT WILL PETER MURRAY OR FRANK HOGG DENTAL LAB BE LIABLE FOR CONSEQUENTIAL DAMAGES EVEN IF PETER MURRAY OR FRANK HOGG DENTAL LAB HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Peter Murray and Frank Hogg Dental Lab also reserve the right to make changes in such material without notice.

TABLE OF CONTENTS

JOB CONTROL PROGRAM

INTRODUCTION

The JOB CONTROL PROGRAM (JCP) reads a text file that contains the necessary input for a program and then supplies this input to the program in the same manner that an operator would have normally entered it from the keyboard. The file containing the input is referred to as the procedure file, and the program receiving the input is referred to as the calling program. A procedure file contains input for such calling programs as FLEXtm, FLEX utility commands, and other development software.

JCP also provides for parameter substitution within the procedure file, special commands to control JCP program flow, and a means for recovery from processing errors. These features allow for commonly used file routines to be written as a generalized procedure that JCP will execute, unattended, simply by entering a single FLEX command.

GETTING THE SYSTEM STARTED

JCP is utilized as a command and the general syntax is:

```
JCP,<filespec>[+]['<parameter list>]
```

where <filespec> is the procedure file to be executed by JCP and defaults to a TXT extension and to the working drive. The <parameter list> is an optional list of parameters that will be substituted in the procedure file. If the "+" option is entered, the echo of JCP lines to the terminal will be turned off.

Examples:

```
+++JCP,1.PROCFL
```

This JCP command call will execute the procedure file PROCFL.TXT found on drive one.

```
+++JCP,1.PROCFL+'2.TEST'2.TEST1
```

This JCP command call is similar to the one used in the previous example, but "2.TEST" and "2.TEST1" will be substituted for parameters one and two wherever used in PROCFL, and the echo of JCP lines will be turned off.

* FLEX is a trademark of Technical Systems Consultants, Inc.

PROCEDURE FILE ORGANIZATION

The procedure file consists of a series of lines. Each line is either a JCP statement (discussed in a later section) or input for a calling program and is terminated with a carriage return (CR). As each line is encountered by JCP, it is first checked for a recognized JCP command to execute. If the line did not contain a JCP command, it is returned as input to the calling program.

Some calling programs accept input on a single character basis (e.g., DELETE, NEWDISK, etc.) and do not require a CR for termination of input. To prevent JCP from returning the CR that terminates the line in a procedure file to this type of calling program, a colon ":" is entered as the first character of the line. Because this type of line is assumed as input for a calling program, there is no attempt to check it for a JCP statement.

Example:

```
DELETE,1.TEST.TXT,1.TEST1.TXT
:YYYY
```

At the beginning of a procedure, FLEX is the initial calling program and JCP will begin execution of the procedure file with the first line. Program flow then continues sequentially until a JCP statement is encountered that will change program flow (e.g., GOTO, CALL, RETURN, IF-ELSE, etc.). The following example procedure file, BACKUP.TXT, is a typical procedure that can be used to make a back-up copy of a diskette:

```
NEWDISK,2
:YY
BACKUP
1
COPY,1,2
END
```

The following JCP command call is used to execute the procedure BACKUP:

```
+++JCP,BACKUP
...START PROCEDURE

+++NEWDISK,2

ARE YOU SURE? Y
SCRATCH DISK IN DRIVE 2? Y
VOLUME NAME? BACKUP

VOLUME NUMBER? 1
```

```
FORMATTING COMPLETE
TOTAL SECTORS = 340
+++COPY,1,2


1.JCP      .TXT TO DRIVE #2     COPIED
1.EDIT     .TXT TO DRIVE #2     COPIED

    COPY COMPLETE

+++END

...PROCEDURE COMPLETED

+++
```

Note that all the lines in this procedure contain input for a calling program, with the exception of the last line (END is a JCP statement). Also note that separate lines are used for the last two responses required for the NEWDISK command and, upon conclusion of the NEWDISK command, the new calling program is again FLEX.

If a line intended as input for a calling program could also be recognized as a JCP statement, a slash "/" is entered as the first character of the line to prevent JCP from executing it.

The procedure file can be created with the FLEX BUILD command or any text editor, such as TSC's, which terminates each line with a CR ($ØD) and no LF ($ØA). JCP also provides a mini-text editor and information on its usage is presented in the section USING THE TEXT EDITOR. Because each line of the procedure file is loaded into the line input buffer used by FLEX, lines must not exceed 128 characters in length.

PARAMETER SUBSTITUTION

Parameter substitution can be used to generalize a procedure file by allowing the input and output specifications of FLEX commands to be represented by parameters. A parameter can be assigned a character string value during the JCP command call, and this value will replace any occurance of the parameter in the procedure file. A parameter can be used as a complete line or any portion of a line and is designated by a "%n"; where "n" is an integer value from one through nine. Parameter substitution is an extremely powerful feature of JCP, and when cleverly used, procedures can be created that are not restricted in use for a specific situation. The following example procedure file, FORMAT.TXT, is a generalized procedure to make a new system diskette:

```
NEWDISK,%1
:YY
SYSTEM
2
COPY,Ø,%1%2
LINK,%1.FLEX2.SYS
%3
```

Character string values can be assigned to parameters during the JCP command call by following the procedure file specification with a list of the desired values. Each value must be preceded by an apostrophe and its position in the list will determine the "n" value assigned. The following JCP command call is used with the procedure file FORMAT to demonstrate:

```
+++JCP,FORMAT'2',.CMD,.SYS'END
```

JCP will display the lines of the procedure file where parameters are used as:

```
NEWDISK,2
COPY,Ø,2,.CMD,.SYS
LINK,2.FLEX2.SYS
END
```

All parameters are initially null at the start of a procedure unless assigned a value during the JCP command call. Any position in the list of values with two apostrophes will also be set to a null. The procedure file FORMAT is used with the following JCP command call to demonstrate:

```
+++JCP,FORMAT'2''END
```

JCP will display the COPY command as:

```
COPY,Ø,2
```

A maximum of nine parameters can be passed to a procedure file, and each one must not exceed 20 characters in length (with the exception of parameter nine which can be 40 characters).

If a slash "/" precedes an apostrophe, the remainder of the JCP command call line will be assigned to the parameter that is due to be set. This can be used for passing parameters from an initial JCP call to chained procedures. The following JCP command call is used with the procedure file FORMAT to demonstrate:

```
+++JCP,FORMAT'1''/'JCP,FORMAT'2''END
```

The value assigned to parameter three becomes:

```
JCP,FORMAT'2''END
```

COMMENTS, LABELS and JCP STATEMENTS

In addition to input for a calling program, a line in a procedure file can be: a comment, label, or JCP statement. As each line of the procedure is processed by JCP, it will be displayed and checked for the form of one of these special lines. If a line is recognized as a comment, label, or JCP statement, JCP will prompt with a "^" on the next line processed. This aids in identifying JCP lines and distinguishes them from input lines for calling programs.

The following sections describe the use of each type of JCP line and in all examples used, it will be assumed that the echo of JCP lines is turned on, unless noted otherwise.

Comment Lines

A line containing an asterisk "*" in the first column, followed by a non-alphanumeric deliminator, is not executed but is displayed. This provides a means for either documenting the procedure file or displaying messages to the terminal during a procedure. The form for a comment line is:

```
* [<comment or message>]
```

where [<comment or message>] is optional.

Example:

```
* THIS IS A COMMENT
*
* OR
*
* PLEASE CHANGE PAPER
* SIZE AND ENTER "CONT"
```

Label Lines

The label line is used in conjunction with the GOTO and CALL statements to mark a search target for a branch. The form for a label line is:

```
. <labelname> [<comment>]
```

Lines containing a label are designated by a period "." in the first column followed by a non-alphanumeric deliminator, followed by the <labelname> which can be up to 20 alphanumeric characters in length. A [<comment>] can be used if separated from <labelname> by a non-alphanumeric deliminator.

Example:

```
. START
```

A line containing a label is not executed but is displayed when encountered during program execution.

JCP Statements

A procedure file can also contain JCP commands. These commands are used in the form of a statement to perform a specific function. This section will describe the form and function of each JCP statement and show examples of their use. The general form for a statement is:

```
<command> <argument> [<comment>]
```

where <command> must begin in the first column and a non-alphanumeric deliminator must be used to separate an <argument>, if required, and, if desired, a [<comment>].

Example:

```
IFSET BREAK   BREAK IF FLAG SET
SET           SET THE FLAG
```

1. GOTO

The GOTO statement provides for unconditional branching to a specified label line. The form of GOTO is:

```
GOTO <labelname> [<comment>]
```

where <labelname> has the form described in the label lines description.

When this statement is encountered, JCP will search for the specified <labelname>. Normal program flow is resumed at the line containing the matching <labelname>. GOTO is useful when used with other statements such as IFSET, IFCLR, IF-ELSE, or ONERROR to skip over sequences of lines.

Example:

```
GOTO LABEL2              Branch to label LABEL2
. LABEL1                 Ignored
COPY,1,2                 Ignored
. LABEL2                 Program resumes here
COPY,Ø,2
```

If a GOTO statement specifies a <labelname> that is not found in a label line, the message "...NO LABEL ON A GOTO - PROCEDURE

ABORTED" will be displayed and JCP will return to FLEX.

2. BREAK, BREAKN and CONT

The BREAK statement causes JCP to halt the processing of the procedure file and return to a manual mode of operation. This statement is useful when operator intervention is necessary, such as changing a diskette or servicing the printer. While in the manual mode, all JCP commands and calling program input can be implemented from the terminal.

Example:

```
* PLEASE CHANGE DISK      Operator prompt
* IN DRIVE 1 AND ENTER
* "CONT"
*
BREAK                     Break occurs here
```

The BREAKN statement is similar to BREAK, except that processing will halt one line after the BREAKN statement is encountered. This statement is useful because it is able to automatically call an applications program where manual data input is desired.

Example:

```
BREAKN
EDIT,TEST,TEST1           Call the Editor and
                          break for manual editing
```

If BREAKN is entered while in the manual mode, JCP will execute the next line of the procedure file and then return to the manual mode. This is useful for single step execution of the procedure file.

When the CONT command is entered, JCP will leave the manual mode and continue processing with the next line of the procedure file. GOTO <labelname>, CALL <labelname>, or RETURN can also be entered to terminate the manual mode and processing will continue at the label line specified or the line following the last executed CALL statement.

Processing can also be halted at any time from the console by entering the ESCAPE key and resumed with either CONT, GOTO, CALL, or RETURN. If a calling program also contains an escape function, JCP may not respond to the ESCAPE key. This is because two break routines are competing for the same ACIA, and if the calling program reads the ACIA before JCP does, the ESCAPE character will be lost. Using a calling program's escape function might cause unpredictable results if serviced, because input that was not accounted for in the procedure file will be expected by the calling program. For this reason, the

escape and pause features of FLEX are disabled during a procedure and restored upon conclusion. For other calling programs that also have an escape function, there is no general solution to the problem other than to disable their escape function.

3. IFSET and IFCLR

The IFSET and IFCLR statements are used to control program execution depending on the status of the program condition code. The form is:

```
IFSET (IFCLR) <command> [<comment>]
```

where <command> can be any JCP command or input for the calling program.

When the IFSET statement is used, JCP will perform the <command> specified if the condition code is set. If the condition code is clear, program execution continues with the line following the IFSET command.

Example:

(Assume the program condition code is clear.)

```
IFSET GOTO L1              The GOTO L1 is ignored
NEWDISK,2                  Program resumes here
:YY
BACKUP
1
GOTO L2
. L1
NEWDISK,1
:YY
BACKUP
1
. L2
```

When the IFCLR statement is used, the <command> specified will be executed if the condition code is clear; otherwise, program execution continues with the next line.

Example:

(Assume the program condition code is clear.)

```
IFCLR GOTO L2              Branch to label L2
BREAKN                     Ignored
EDIT,%1,%2                 Ignored
S                          Ignored
. L2                       Program resumes here
ASMB,%2,+LS
```

4. SET and CLR

The SET and CLR statements are used to change the status of the program condition code that is used with the IFSET and IFCLR statements.

The condition code can either be set to a logic "1" using the SET statement or cleared to a logic "Ø" using the CLR statement and is initially clear at the start of a procedure.

Example:

(Assume the program condition code is initially clear.)

```
. L1
*           INSERT DISK IN DRIVE 2
BREAK       AND ENTER "CONT"
NEWDISK,2
:YY
BACKUP
1
COPY,1,2
IFSET GOTO L2
SET
GOTO L1
. L2
CLR
```

This set of instructions will format drive two, copy drive one to drive two, then repeat the process one more time.

The program condition code can also be set and cleared from calling programs such as BASIC (PEEK and POKE statements). Information on the memory location of the condition code can be found in APPENDIX D: SYSTEM ADAPTATIONS.

5. %n=

The %n= statement is used to assign a string value to a parameter. The form is:

```
%n=<string>
```

where "n" designates the parameter to be replaced with <string>.

Example:

```
%1=PROG
%2=,+LS
%1=ASMB,%1%2              Sets %1 to "ASMB,PROG,+LS"
```

All parameters are initially null unless assigned a value

during the JCP command call and can be reset to a null by entering "%n=" followed by a CR.

6. IF, IFN and ELSE

The IF and IFN statements are branch commands that are conditioned by the result of a comparison. The form is:

```
IF (IFN) %n=<string>
 .
ELSE
```

where "n" designates the parameter that is to be compared with <string>.

When the IF statement is used, program execution continues with the line following the IF statement when the specified parameter is equal to <string>; otherwise, a branch to the next ELSE occurs.

Example:

```
%1=ASM
 .
. ASM
IF %1=ASM                    True; don't branch to ELSE
%1=%2                        Program continues here
ASMB,%1,+LS
GOTO NEXT
ELSE
IF %1=COMPIL
```

When the IFN statement is used, program execution continues with the line following the IFN statement when the specified parameter is not equal to <string>; otherwise, a branch to the next ELSE occurs.

Example:

```
%1=PROG1
 .
%2=PROG1
 .
IFN %1=%2                    False; branch to next ELSE
%1=%2                        Ignored
SET                          Ignored
ELSE                         Program resumes here
%2=%2.BIN
```

If an ELSE statement does not follow an IF or IFN statement, and a branch is required, the message "...NO ELSE ON AN IF OR IFN - PROCEDURE ABORTED" will be displayed and JCP will return to FLEX.

7. CALL and RETURN

The CALL statement is used to transfer program control to a sequence of lines that need to be executed more than once during a procedure. The form of a CALL statement is:

```
CALL <labelname> [<comment>]
```

Upon execution of CALL, JCP performs a branch to the line containing <labelname> where normal program flow continues until a RETURN statement is executed. RETURN will cause program execution to continue at the line following the last executed CALL statement.

Example:

```
%1=1
CALL COPY
%1=2
CALL COPY
END
*
* SUBROUTINE COPY SYSTEM DISK
*
. COPY
NEWDISK,%1
SYSTEM
2
COPY,Ø,%1
LINK,%1.FLEX2.SYS
RETURN
```

This procedure will make a back-up copy of the system diskette on both drives one and two.

CALL statements can be nested seven levels in depth. If this value is exceeded, or a label is not found on a CALL, or a RETURN statement is encountered without a prior CALL, a fatal error will occur and JCP will display the appropriate message and return to FLEX.

8. ONERROR

The ONERROR statement provides a means to recover from operating system errors. The form for ONERROR is:

```
ONERROR <command> [<comment>]
```

where <command> can be any JCP or FLEX command.

The ONERROR statement does not take any action at the time it is executed, but specifies a <command> to be executed should a FLEX error occur later during processing. When this statement

is used and an error occurs, <command> will be executed and displayed with the message "...TRAPPED BY ONERROR". If an error occurs and no ONERROR statement has been executed, the message "...ERROR TRAP NOT SPECIFIED - PROCEDURE ABORTED" will be displayed and JCP will return to FLEX. Any FLEX error that should occur while in the manual mode will not cause an ONERROR statement to be executed when the processing of the procedure file is resumed.

Example:

```
. COPY
ONERROR GOTO L1
. L1
NEWDISK,%2
:YY
BACKUP
1
COPY,%1,%2
ONERROR BREAK
RETURN
```

In this example the first ONERROR statement will cause a branch to label L1 if an error occurs during the NEWDISK or COPY command. If the copy is successful, the second ONERROR statement will cause a BREAK if an error should occur later in the procedure.

Care should be taken in specifying <command>. If an unrecognized JCP or FLEX command is entered, a dead loop will occur.

9. END

The END statement is used to stop execution of the procedure file and return control to the operating system. When this statement is encountered, the message "...PROCEDURE COMPLETED" will be displayed and JCP will return to FLEX. END is optional; if it is not used and the end of the procedure file is reached, the message "...EOF REACHED - PROCEDURE TERMINATED" will be displayed and JCP will return to FLEX. END can also be used more than once in a procedure file.

Example:

```
ONERROR END
* CHECK FOR A NULL PARAMETER
IF %1=(CR)
END
ELSE
ASMB,%1,+LS
GET,%1
END
```

A procedure can also be stopped at any time during execution by forcing a break with the ESCAPE key and entering END.

1Ø. EON and EOFF

The EON and EOFF statements are used to control the display of JCP comment, label, and statement lines.

When the EON statement is used, all subsequent lines of the procedure file will be echoed to the terminal. When the EOFF statement is used, only subsequent lines that contain input for calling programs will be displayed. EOFF does not affect statements that are trapped by the ONERROR statement, and they will always be displayed.

Example:

(Assume EON is initially in effect.)

```
EOFF
* COPY ROUTINE
ONERROR GOTO NEXT
COPY,1,2
. L1
:N
GOTO L1
. NEXT
EON
BREAK
```

This sequence of statements will supply an unknown number of "N" responses to the COPY command. The EON and EOFF statements are used to clarify the output, and the only lines that will be displayed are:

```
EOFF
COPY,1,2
:N
BREAK
```

The "GOTO NEXT", which is specified by the ONERROR statement, will also be displayed when the unrecognized "N" response is returned to FLEX and the error is trapped.

If neither EON or EOFF is specified in the procedure file, the default assignment can be specified during the JCP command call. An optional switch can be entered and its form is:

```
JCP,<filespec>[+]['<parameter list>]
```

When the plus "+" is present, the default assignment is EOFF; otherwise, EON is used.

11. +

The "+" statement is used to implement FLEX commands from user programs such as EDIT, BASIC, SORT or any calling program that uses FLEX's character or line input routine. The form is:

```
+<string>
```

where <string> is a FLEX command and immediately follows "+".

When this statement is encountered, JCP will send <string> to FLEX for execution as a command. Upon conclusion of the FLEX command invoked, control will return to the calling program.

Example:

```
EDIT,T1,T2
36Ø
+SETPlE
P22Ø
+CLRPlE
```

This example represents a portion of an editing procedure that will produce a partial listing to the printer. Both calls to FLEX are user defined commands to control terminal echo to the printer.

If the "+" statement calls a program that has an input portion, "+" must not be used again until there is a return from the first FLEX command call. If this is attempted, the message "...+ IGNORED" will be displayed.

Caution must be taken when using the "+" statement. A calling program must not invoke a FLEX command that would use the same area of memory. Another area of caution concerns the closing of all open files. JCP will prevent the DOS portion of FLEX from closing all open files during the execution of the "+" statement; however, if the calling program has any open files, the FLEX command invoked must not attempt to close all open files or use an active file control block.

USING THE TEXT EDITOR

INTRODUCTION

JCP provides a mini-text editor to aid in quickly creating new procedure files that can be immediately executed and/or saved on disk to be used at a later time. The editor contains only a brief set of commands for manipulating the text file; therefore, its use is not recommended for creating large files where a more extensive text editor should be used.

GETTING THE EDITOR STARTED

The editor is accessed from FLEX as a command, and its syntax is:

```
JCP[,<filespec 1>]
```

If <filespec 1> does not exist on the disk, a new file will be created as specified by <filespec 1>. The default extension is TXT, and the default drive is the working drive.

The editor can also be called by entering:

```
JCP(CR)
```

This command will call the editor, but will not create a disk file for saving the new text. This entry is useful for creating short procedures that JCP can immediately execute.

After entering either one of these commands, the editor will respond with:

```
EDITOR READY
  >
```

EDITOR OPERATION

The editor has two modes of operation--the insert and command mode. Upon calling the editor, the insert mode is in effect and the editor will prompt with ">" to indicate that text input is ready to be accepted. Each line entered is terminated with a carriage return (CR) and can be edited or deleted using the normal line editing features of FLEX. If a "#" is entered as the first character of a line, the editor will leave the insert mode and prompt with "#" to indicate that the command mode is in effect.

EDITOR COMMANDS

This section will describe the various editor commands available for manipulating the text file. All commands are entered while in the command mode with the exception of "#".

COMMAND	DESCRIPTION
P	Print the current line.
P!	Print from the current line to the last line and make the last line the current line.
N	Make the next line the current line and print it. A carriage return can also be entered in place of N.
-	Make the previous line the current line and print it.
T	Make the first line the current line and print it.
N!	Make the last line the current line and print it.
D	Delete the current line; make the next line the current line and print it.
=<string>	Replace the current line with <string> and print it.
I	Leave the command mode and begin line input after the current line.
#	Leave the insert mode; return to the command mode; make the last inserted line the current line and print it.

EXITING THE EDITOR

The following commands are used to exit from the editor and are entered while in the command mode of operation:

```
FLEX
```

This command will write and close <filespec 1> if it was specified when the editor was called. Control will then return to FLEX.

```
JCP[,][<filespec 2>][+]['<parameter list>]
```

This command will write and close <filespec 1> if it was specified when the editor was called (RUN can be used in place of JCP). If <filespec 2> is specified, JCP will use it as the procedure file and begin execution using an optional list of parameters. The default extension is TXT, and the default drive is the working drive. If <filespec 2> is not specified, JCP will use the text created with the editor as its procedure file to execute. The echo of JCP lines will default to EOFF if the "+" is entered.

The following examples will help to clarify the use of this command:

```
JCP
RUN,+'Ø'2
JCP,SORTJCP
```

The first example will execute the procedure file created with the editor. The second example is similar to the first example, but this time the echo is turned off and values are assigned to parameters. The last example will execute the procedure file SORTJCP.TXT.

TEXT EDITING EXAMPLE

This example will demonstrate some of the editor's commands. The following JCP command call will be used to call the editor and save the new procedure file on drive one as EXAMPLE.TXT:

```
+++JCP,1.EXAMPLE

EDITOR READY
  >* IN THE INSERT MODE UPON
  >* ENTERING THE EDITOR
  >*
  >BASIC
  >LOAD "1.SORT"
  >RUN
  >*
  >* DATA FOR BASIC PROGRAM
```

```
  >*
  >2.DATA
  >KYFL
  >2.DATA.KEY
  >*
  >* PROGRAM FINISHED AT
  >* THIS POINT
  >*
  >END
  >#  LEAVE THE INSERT MODE
  >END
#T
  >* IN THE INSERT MODE UPON
#D
  >* ENTERING THE EDITOR
#=* PROCEDURE TO RUN BASIC
  >* PROCEDURE TO RUN BASIC
#I
  >* PROG TO SORT FILE 2.DATA
  >* INTO KEYFILE 2.DATA.KEY
  >#
  >* INTO KEYFILE 2.DATA.KEY
#P
  >* INTO KEYFILE 2.DATA.KEY
#N
  >*
#T
  >* PROCEDURE TO RUN BASIC
#P!
  >* PROCEDURE TO RUN BASIC
  >* PROG TO SORT FILE 2.DATA
  >* INTO KEYFILE 2.DATA.KEY
  >*
  >BASIC
  >LOAD "1.SORT"
  >RUN
  >*
  >* DATA FOR BASIC PROGRAM
  >*
  >2.DATA
  >KYFL
  >2.DATA.KEY
  >*
  >* PROGRAM FINISHED AT
  >* THIS POINT
  >*
  >END
#RUN
```

APPENDIX A: JCP COMMAND SUMMARY

GETTING THE SYSTEM STARTED:

JCP,<filespec>[+]['<parameter list>]

Execute <filespec> with an optional list of parameters and turn off the echo of JCP lines if a "+" is entered.

COMMAND SUMMARY:

COMMAND	DESCRIPTION	PAGE
*	Comment.	6
. <labelname>	Label line.	6
GOTO <labelname>	Branch to <labelname> and continue execution.	7
BREAK	Suspend processing.	8
BREAKN	Suspend processing after the next line.	8
CONT	Continue processing.	8
IFSET <command>	Execute <command> if the condition code is set.	9
IFCLR <command>	Execute <command> if the condition code is clear.	9
SET	Set the condition code.	10
CLR	Clear the condition code.	10
%n=<string>	Replace the specified parameter with <string>.	10
IF %n=<string> . ELSE	If the specified parameter equals <string>, program execution continues with the next line; otherwise, branch to the next ELSE.	11
IFN %n=<string> . ELSE	If the specified parameter is not equal to <string>, program execution continues with the next line; otherwise, branch to the next ELSE.	11
CALL <labelname>	Branch to <labelname> and continue execution until a RETURN is encountered, then continue execution with the line following the last CALL.	12
RETURN	Return to the line following the last CALL.	12
ONERROR <command>	Execute <command> in the event of an error.	12
END	End procedure.	13
EON	Turn on the echo of JCP lines.	14
EOFF	Turn off the echo of JCP lines.	14
+<string>	Deliver <string> to FLEX for execution, then continue executing with the line following +<string>.	15

APPENDIX B: EDITOR COMMAND SUMMARY

GETTING THE EDITOR STARTED:

JCP[,<filespec 1>]

Call the editor and if <filespec 1> is entered and does not exist as a disk file, a new file will be created as specified by <filespec 1>.

COMMAND SUMMARY:

COMMAND	DESCRIPTION
P	Print the current line.
P!	Print from the current line to the last line and make the last line the current line.
N or (CR)	Make the next line the current line and print it.
-	Make the previous line the current line and print it.
T	Make the first line the current line and print it.
N!	Make the last line the current line and print it.
D	Delete the current line, make the next line the current line and print it.
=<string>	Replace the current line with <string> and print it.
I	Leave the command mode and begin line input after the current line.
#	Leave the insert mode; return to the command mode; make the last inserted line the current line and print it.

EXITING THE EDITOR:

FLEX

Write and close <filespec 1> if it was specified when the editor was called. Then return control to FLEX.

RUN or JCP[,][<filespec 2>][+]['<parameter list>]

Write and close <filespec 1> if it was specified when the editor was called. If <filespec 2> is not entered, execute the text created with the editor as a procedure file with an optional list of parameters. If <filespec 2> is entered, it will be used as the procedure file. The echo of JCP lines will default to off if the "+" is entered.

APPENDIX C: PROGRAM CHARACTERISTICS

1. A file intended for use with the FLEX EXEC command can contain more than one JCP command call. Upon conclusion of each procedure, the next line of the EXEC file will be executed, unless there was an untrapped error in the procedure which will cause the EXEC command to abort. The same is true for a procedure file; it can contain more than one EXEC command. Upon conclusion of each EXEC file, the next line of the procedure file will be executed. JCP can also be used as a FLEX command within a procedure file to chain another procedure, but upon conclusion of the chained procedure, JCP will not return to the previous procedure.

2. Some versions of BASIC provide a means for executing a FLEX command either from the command mode, or as a statement in a BASIC program. This FLEX command can be a JCP call; upon conclusion of a procedure, control will return to BASIC. The procedure can contain FLEX commands that will load into the same area of memory as BASIC if it first saves the BASIC pointers and program area as temporary files; upon conclusion of the procedure, BASIC and the temporary files are loaded into memory. Using a procedure in this manner provides the programmer with an extremely powerful tool. A BASIC program can now use JCP to control the execution of a sequence of FLEX commands. For example, a procedure could be used to execute the FLEX sort utility on a large set of data and then run a PASCAL program to compute various statistical values.

   If a BASIC program calls a procedure that contains an EXEC command, the values at addresses $AC43-$AC44 must also be saved and then restored upon conclusion of the procedure.

3. If a dead loop should occur because of programming logic in a procedure, the ESCAPE key can be entered to force a break.

4. Do not use the FLEX SAVE command on any memory resident portion of FLEX. JCP makes patches to the DOS portion of FLEX at the start of a procedure; to copy this portion of memory will not reflect the proper code.

5. When in the manual mode of operation and a FLEX command is invoked that accepts input on a single character basis, do not forget to enter a colon ":" as the first character. To terminate input, a carriage return will also have to be entered, but it will not be returned to the calling

program.

6. Some FLEX commands, which accept input on a single character basis and do not use the FLEX line input routine (INBUFF), require that the FLEX line buffer stay intact. Although JCP loads the lines of the procedure file into this buffer for processing, it will always be restored upon return of each line of input to a calling program of this type.

   Since the line input buffer is not restored for calling programs that use the INBUFF routine for input, multiple FLEX commands should not be used on a single line.

7. When loading a procedure file from disk, JCP will perform a memory end check; however, this check is not made when the editor is used to create the text, nor does JCP check for buffer overflow on lines, parameters, or labels. To exceed the storage limits set might cause unpredictable results.

APPENDIX D: SYSTEM ADAPTATIONS

SYSTEM REQUIREMENTS

JCP is designed to run with Technical Systems Consultants' FLEXtm 2.Ø Disk Operating System. The only special requirement is that the DOS portion of FLEX be located in RAM and not in ROM because JCP makes patches to DOS.

The memory requirements are approximately 2K bytes for the JCP command processor, plus storage for the procedure file and program variables. For applications where procedures are less than a 1ØØ lines, a 4K block of memory will provide more than enough room for the command processor and a procedure file of approximately 1.6K bytes in size. The version of JCP that is supplied will require 4k bytes of memory beginning at $7ØØØ. If this configuration will not work on your system, then refer to the next section for information on reassembly of JCP for a different location (see EQU directive, PART2).

Calling programs that do not use the input routines of FLEX for keyboard input will not work with JCP and must be changed to use either the line input routine (INBUFF) or the character input routine (GETCHR).

MAKING CHANGES TO JCP

It is recommended that the process of reassembly be used for any reconfiguration or modification made to JCP. The necessary source code files JCP.TXT, EDIT.TXT, and JCPEQU.TXT are supplied, and this section will describe the various assembler directives that affect JCP program organization and operating characteristics. Provided that JCP is already running on your system, the procedure file GENJCP.TXT, which is also supplied, can be used to easily reconfigure JCP. For information on how to use this procedure, refer to the next section.

The following directives are listed as they appear in the LIB file JCPEQU.TXT and can be changed to suit a particular system configuration:

PLIMIT EQU 9
: This value determines how many parameters can be used and it must not exceed nine.

PSIZE EQU 2Ø
: This value sets the size of the parameters. To determine the total amount of storage used by the parameters, use the following formula: STORAGE=(PSIZE+1)*PLIMIT+ENDBF

* FLEX is a trademark of Technical Systems Consultants, Inc.

```
ENDBF    EQU   2Ø
```
The size of the last parameter can be extended in length by the value of ENDBF.

```
BUFSZ1   EQU   2Ø
```
This value sets the size of the buffer used for holding <command> and the remainder of the line in an ONERROR statement.

```
BUFSZ2   EQU   2Ø
```
This value sets the size of the buffer used for holding either a label that is the target of a branch, or a parameter that is used in a comparison. This buffer must be as large as any parameter that will be used in a comparison.

```
BUFSZ3   EQU   48
```
This value sets the size of the buffer used for saving the FLEX line input buffer. The size of this buffer only needs to be as large as a line that contains a command call to a program such as PR or SORT, which accept input on a character basis and require that the line input buffer stay intact.

```
CLIMIT   EQU   7
```
This value determines the nesting limit of CALL statements.

```
PDELIM   EQU   ''
```
This value determines the field separator character that is used when parameters are entered during the JCP command call.

```
DELCHR   EQU   '/
```
This value determines the special line deliminator character that is used to prevent JCP from processing a line as one of its statements.

```
DLCHR1   EQU   '/
```
This value determines the character used to precede a parameter deliminator when it is desired to assign the remainder of a JCP command call line to the parameter that is due to be set.

```
DOSPR    EQU   '+
```
This value determines the character that is used when accessing FLEX from the input portion of a calling program.

```
BRKCHR   EQU   $1B
```
This value determines the break character which is currently set as the ASCII ESC character.

```
PART1    EQU   $A1ØØ
```
This portion of the program requires approximately .6K bytes of memory and contains the routines for clearing the

storage area, reading the procedure file, initializing the parameters, and making patches to FLEX. It is not necessary to preserve this part of the program during a procedure; therefore, it can be assembled in an area of memory that is used by calling programs, and is currently located at the beginning of the Utility Command Space.

PART2 EQU $7000

This portion of the program requires approximately 2K bytes of memory and contains the JCP command processor and the routines for supplying the input to calling programs and restoring FLEX upon conclusion of a procedure. This portion of JCP must be preserved during a procedure; therefore, it must be assembled in an area of memory that will not be used by a calling program.

If JCP is assembled at a location that is below FLEX's memory end pointer (MEMEND), then JCP will adjust MEMEND to point just below PART2 and restore it upon conclusion of a procedure. Note: this adjustment will not protect PART2 from calling programs that do not use MEMEND to check where the end of memory is. In addition, the EXEC command will not work if called from a procedure; however, an EXEC file can contain a JCP call, provided that PART2 or the procedure file will not destroy the EXEC command.

For best performance it is recommended that JCP be located above the DOS portion of FLEX (e.g. $C000, $D000) or at any location above MEMEND. This will insure the protection of PART2 and not decrease the workspace size.

CMDSTR EQU PART2+$0FFF

This value determines the end of memory address used when loading the procedure file, and is currently set to use the remainder of the 4K block of memory that PART2 is assembled in.

The remaining assembler directives are found in the source code file JCP.TXT:

```
        ORG   *
EDIT    EQU   *
```

This portion of the program contains the command processor for the editor and occupies approximately .6K bytes of memory. It is not necessary to preserve the editor once a procedure begins; therefore, it can be assembled in an area of memory that is used by calling programs, and is currently assembled to follow PART1.

The editor is supplied as a separate program, EDIT.TXT, for convenience and clarity, and is assembled with JCP.TXT as a LIB file. If EDIT.TXT is assembled separately, the external EQU's used for calls to JCP routines must be set to the proper address.

```
        ORG   *
BEGSTR  EQU   *
```

This directive establishes the beginning of JCP's variable storage area which is currently assembled to follow PART2.

```
IFFLG3  RMB   1
```

This byte of memory is used as the flag for the SET and CLR statements, and its address can be determined by assembly of the source code. IFFLG3 can be referenced by outside programs and will be considered set if it contains a non-zero value.

```
        ORG   *
CBFFR1  EQU   *
```

The procedure file is loaded into memory beginning at the address this directive sets, and is currently set to follow the variable storage area. If this value is changed, do not forget to change the end of memory pointer address (CMDSTR) accordingly.

ASSEMBLY PROCEDURE

The procedure GENJCP.TXT will automate the process of editing and assembling a new version of JCP. Several parameters can be entered that will be substituted for various assembler directives, but in most cases it will only be necessary to relocate PART2. This procedure will require TSC's editor and assembler, and its output will be the command file JCP1.CMD, which can be renamed and copied to the system diskette.

To use this procedure you will first need to format a diskette and copy the files JCPEQU.TXT, JCP.TXT, EDIT.TXT, and GENJCP.TXT. Now assign the work drive and enter the following JCP command call:

```
JCP,GENJCP[+]['P#1'P#2'P#3]
```

where parameters 1-3 are as follows:

```
P#1 = PART2
P#2 = PLIMIT
P#3 = PSIZE
```

The following JCP command call will relocate PART2 to begin at $C000, and increase the size of the parameters to 25:

```
+++JCP,GENJCP'C000''25
```

If you are not able to use GENJCP or have to make changes manually to one of the source files, then the following FLEX command will assemble the new version of JCP as a command:

```
ASMB,JCP,JCP.CMD,+LS
```

It is required that the files JCPEQU.TXT and EDIT.TXT retain these names because they are both LIB files. It is also necessary that the three files JCPEQU.TXT, JCP.TXT, and EDIT.TXT be located on the working drive.

APPENDIX E: PROCEDURE FILE EXAMPLES

Some of the procedures illustrated are also supplied on the diskette with JCP, and the user is encouraged to study the coding techniques they demonstrate.

To help clarify some of the lines, a carriage return entry is denoted by "(CR)".

EXAMPLE 1

This example will demonstrate how a short procedure can be easily created with JCP's text editor and immediately executed. The procedure will perform various disk utilities and will not be saved as a disk file. The following JCP command is used to call the editor:

```
+++JCP

EDITOR READY
  >ONERROR BREAK
  >ASM,W=1
  >EXEC,Ø.FORMAT2
  >O,2.TMP1,PR,APXE
  >1,1
  >PRINT,2.TMP1
  >O,2.TMP2,TYPOS,2.TMP1.OUT
  >PRINT,2.TMP2
  >END
  >#
  >END
#RUN

...START PROCEDURE

+++ONERROR BREAK
^ASM,W=1

NOT FOUND
+++BREAK    ...TRAPPED BY ONERROR
^ASN,W=1

+++CONT
^EXEC,Ø.FORMAT2

+++O,2.TMP1,PR,APXE

PAGE LIMITS? 1,1

+++PRINT,2.TMP1
```

```
+++O,2.TMP2,TYPOS,2.TMP1.OUT

+++PRINT,2.TMP2

+++END

...PROCEDURE COMPLETED

+++
```

Note that it was a simple matter to manually enter the ASN command when the error was trapped, and then enter CONT in order for JCP to continue with the next line of the procedure.

EXAMPLE 2

This procedure file will show how parameters can be used to generalize the input and output specifications of a series of disk utilities.

The procedure file WORDCNT.TXT will link together PR, TYPOS, and CSORT to create a file containing an alphabetical list of the words in a text file that is coded for use with the word processor. A BASIC program will then list this file to the printer with a four column format.

The general syntax of the JCP command call for the WORDCNT procedure is:

```
JCP,WORDCNT[+]'P#1'P#2
```

where P#1 designates the working drive to be used during the procedure, and P#2 designates the file specification for the coded text file to be processed. The default extension is TXT, and the default drive is the current working drive. A JCP command call using the the procedure file WORDCNT follows:

```
+++JCP,WORDCNT'1'APXA
```

This command line will use drive one as the working drive during the procedure, and process the coded text file APXA.TXT, which is located on the current working drive.

If it is desired to process more than one file, then the EXEC command can be used with an EXEC file that contains several calls to JCP. The following EXEC file will demonstrate this:

```
JCP,WORDCNT'1'APXA
JCP,WORDCNT'1'APXB
JCP,WORDCNT'1'APXC
```

This EXEC file will execute the WORDCNT procedure three times using the coded text files APXA.TXT, APXB.TXT, and APXC.TXT.

The following is a listing of the procedure file WORDCNT.TXT:

```
ONERROR BREAK
*
*  PROCESS THE TEXT FILE
*  WITH THE WORD PROCESSOR
*
O,%1.TMP1,PR,%2
(CR)
*
*  CALL THE TYPOS UTILITY
*
O,%1.TMP2,TYPOS,%1.TMP1.OUT
```

```
PDEL,%1.TMP1.OUT
:Y
*
*  CALL THE SORT UTILITY
*
CSORT,%1.TMP2.OUT,%1.TMP3,+(1)6-16,+(1)6-*,(1)3-5
PDEL,%1.TMP2.OUT
:Y
*
*  LOAD BASIC AND ENTER THE PROGRAM
*
BASIC
50  OPEN "0.PRINT" AS 0:C1=0:ONERROR GOTO 500
100 INPUT "ENTER FILESPEC",L1$:OPEN OLD L1$ AS 1
200 INPUT #1,L1$:PRINT #0,TAB(C1*20);L1$;
300 IF C1=3 THEN C1=0:PRINT #0:GOTO 200
400 C1=C1+1:GOTO 200
500 PRINT #0:END
*
*  RUN THE PROGRAM AND ENTER THE DATA
*
RUN
%1.TMP3.TXT
*
*  RETURN TO FLEX AND END PROCEDURE
*
FLEX
PDEL,%1.TMP3.TXT
:Y
END
```

EXAMPLE 3

In the previous example it was shown how the EXEC command could be used to execute the WORDCNT procedure several times. The procedure file BUILDEX.TXT will use the BUILD command to automatically create the EXEC file that was necessary, and then invoke the EXEC command using this new file.

The general syntax of the JCP command call for the BUILDEX procedure is:

```
JCP,BUILDEX[+]'P#1['P#2-P#8]
```

where P#1 designates the working drive to be used during the WORDCNT procedure, and P#2-P#8 designate the file specifications for the coded text files to be processed. The default extension is TXT, and the default drive is the current working drive. The procedure file WORDCNT must be located on the current working drive. The following JCP command call is used with the BUILDEX procedure to demonstrate:

```
+++JCP,BUILDEX+'1'APXA'APXB'APXC
```

This command line will create an EXEC file, which contains three calls to the WORDCNT procedure; this EXEC file will then be executed. WORDCNT and the three coded text files to be processed APXA.TXT, APXB.TXT, and APXC.TXT are located on the current working drive, and drive one will be used as the working drive during the WORDCNT procedure.

The following is a listing of the procedure file BUILDEX.TXT:

```
*
*  CHECK FOR NULL P#2
*
IF %2=(CR)
EON
*  NO FILES ENTERED
END
ELSE
*
*  TRY AND DELETE OLD EXEC FILE
*
ONERROR CONT
PDEL,JCPEX
:Y
*
*  SET TRAP & CALL THE BUILD COMMAND
*
ONERROR END
BUILD,JCPEX
*
*  CHECK P#2 FOR A NULL VALUE
*  IF NULL, QUIT BUILD AND EXEC FILE
```

```
*  IF NOT NULL, ENTER LINE & LOOP
*
. LOOP
IFN %2=(CR)
CALL ENTER
GOTO LOOP
ELSE
%2=#
%2
EXEC,JCPEX
*
*  ROUTINE TO ENTER LINES
*  AND SHIFT PARAMETERS
*
. ENTER
%9=JCP,WORDCNT+'%1'%2
%9
%2=%3
%3=%4
%4=%5
%5=%6
%6=%7
%7=%8
%8=(CR)
RETURN
```

EXAMPLE 4

Rather than use the EXEC command to process more than one file with the WORDCNT procedure, a better method is to use a modified form of WORDCNT, which contains some added JCP control statements. The procedure file WORDCNT1 will perform the same function as the WORDCNT procedure; however, it will be able to process more than one coded text file by using the IF-ELSE and CALL-RETURN statements.

The general syntax of the JCP command call for the WORDCNT1 procedure is:

```
JCP,WORDCNT1[+]'P#1['P#2-P#9]
```

where P#1 designates the working drive to be used during the procedure, and the remaining parameters are used to designate the file specifications for the coded text files to be processed. The default extension is TXT, and the default drive is the current working drive. The following JCP command call will use the procedure file WORDCNT1 to demonstrate:

```
+++JCP,WORDCNT1'1'APXA'APXB'APXC
```

This command line will use drive one as the working drive during the procedure, and will process the coded text files APXA.TXT, APXB.TXT, and APXC.TXT, which are located on the current working drive.

The following is a listing of the procedure file WORDCNT1.TXT:

```
ONERROR BREAK
*
*  CHECK P#2 FOR A NULL VALUE
*  IF NULL, END PROCEDURE
*  IF NOT NULL, CALL WORDCNT,
*  SHIFT PARAMETERS, AND LOOP
*
. LOOP
IF %2=(CR)
END
ELSE
CALL WORDCNT
%2=%3
%3=%4
%4=%5
%5=%6
%6=%7
%7=%8
%8=%9
%9=(CR)
GOTO LOOP
*
*  WORDCNT PROCEDURE CALLED
```

```
*  AS A SUBROUTINE
*
. WORDCNT
*
*  PROCESS THE TEXT FILE
*  WITH THE WORD PROCESSOR
*
O,%1.TMP1,PR,%2
(CR)
*
*  CALL THE TYPOS UTILITY
*
O,%1.TMP2,TYPOS,%1.TMP1.OUT
PDEL,%1.TMP1.OUT
:Y
*
*  CALL THE SORT UTILITY
*
CSORT,%1.TMP2.OUT,%1.TMP3,+(1)6-16,+(1)6-*,(1)3-5
PDEL,%1.TMP2.OUT
:Y
*
*  LOAD BASIC AND ENTER THE PROGRAM
*
BASIC
50  OPEN "Ø.PRINT" AS Ø:C1=Ø:ONERROR GOTO 5ØØ
1ØØ INPUT "ENTER FILESPEC",L1$:OPEN OLD L1$ AS 1
2ØØ INPUT #1,L1$:PRINT #Ø,TAB(C1*2Ø);L1$;
3ØØ IF C1=3 THEN C1=Ø:PRINT #Ø:GOTO 2ØØ
4ØØ C1=C1+1:GOTO 2ØØ
5ØØ PRINT #Ø:END
*
*  RUN THE PROGRAM AND ENTER THE DATA
*
RUN
%1.TMP3.TXT
*
*  RETURN TO FLEX
*
FLEX
PDEL,%1.TMP3.TXT
:Y
RETURN
```

EXAMPLE 5

TSC's new BASIC provides a statement that will access a FLEX command from a program. This command can be a call to JCP; upon conclusion of the procedure invoked, control will return to the next statement in the BASIC program. This example will demonstrate a technique that will allow a procedure to contain calls to programs that use the same memory as BASIC. The following BASIC statements might be a segment of a program that uses the procedure file JCPSORT.TXT to sort a data file:

```
500 INPUT "SORT FILE 1, 2, OR 3",F$
510 REM
530 REM...AND SOMEWHERE DOWN THE ROAD THIS
540 REM...NEXT STATEMENT MIGHT BE EXECUTED
550 EXEC,"JCP,JCPSORT+'" + F$
560 REM...PROGRAM WILL RESUME HERE UPON
570 REM...CONCLUSION OF THE PROCEDURE
```

JCPSORT uses parameter one to designate the data file and parameter file for the PSORT command, which is used in the procedure.

The following is a listing of the procedure file JCPSORT.TXT:

```
*  PROCEDURE CALLED FROM A BASIC
*  PROGRAM TO USE PSORT ON A DATA FILE
*
ONERROR GOTO RESTORE
*
*  SAVE BASIC PROGRAM AND VARIABLES
*
SAVE,TMP1,0000,00FF
SAVE,TMP2,3600,5000
*
*  DO THE SORT
*
PSORT,PARMFL%1,DATAFL%1
*
*  LOAD BASIC, REPLACE PROG. & VAR.
*
. RESTORE
ONERROR BREAK
GET,0.BASIC.CMD
GET,TMP1
GET,TMP2
PDEL,TMP1,TMP2
:YY
*
*  RETURN TO BASIC PROGRAM
*
END
```

EXAMPLE 6

Another application of JCP is in the development of programs that are written in a language that is compiled. The major advantage of using a compiler over an interpreter is a faster program, but the disadvantage is the extra time required for program development. This is because some compilers use a multi-step process to generate the object code, and it is not uncommon for programs of considerable size to require several minutes between each step.

The procedure file COMPILE.TXT will show how these steps can be automated when using the SOFTWARE DYNAMICS BASIC COMPILER and MAL ASSEMBLER. SD's BASIC is a two-pass compiler that produces extremely fast object code. To use this package, the BASIC source code must first be compiled. Next, the output from the compiler is assembled with the MAL ASSEMBLER. Finally, the assembler's output is combined with the runtime package and the program is executed. Note that COMPILE will not reduce the time required for the process, but will enable the operator to enter a single command that will eliminate intervention between the steps.

The general syntax of the JCP command call for the COMPILE procedure is:

```
JCP,COMPILE[+]'P#1['P#2]
```

where P#1 designates the BASIC source file to be compiled. This file is entered without its extension, which must be a TXT, and the drive specification used for all input and output will default to the working drive if not entered with the file name. P#2 is optional; it designates a file specification that will be used with the FLEX I command to supply keyboard input to the BASIC program. If input is not required by the program, then it is only necessary to enter P#1. The following JCP command call will use the procedure COMPILE to demonstrate:

```
+++JCP,COMPILE'SIMOEQU'NODEFL1
```

This command call would perform the necessary steps to compile the BASIC source code file, SIMOEQU.TXT, found on the working drive. Upon conclusion of the procedure, the runtime package and binary file, SIMOEQU.BIN, will be loaded into memory and executed using the file NODEFL1.TXT as input for the program. It is not necessary to use the I command for supplying data to the program. If preferred, the test data can be added to the procedure for automatic input, or the BREAK or BREAKN statements can be used for manual input.

The following is a listing of the procedure file COMPILE.TXT:

```
*
*  TRY AND DELETE OLD FILES
*
ONERROR CONT
PDEL,%1.ASM,%1.BIN
:YY
*
*  SET TRAP, CALL THE COMPILER AND
*  ENTER INPUT AND OUTPUT SPECS.
*
ONERROR BREAK
SDCOM
%1
%1
*
*  CALL THE ASSEMBLER AND ENTER
*  THE INPUT AND OUTPUT SPECS.
*
SDASM
%1
(CR)
%1
(CR)
*
*  FINISHED WITH THE COMPILER'S
*  OUTPUT, SO DELETE IT.
*
PDEL,%1.ASM
:Y
*
*  SET P#2 IF NOT NULL
*
IFN %2=(CR)
%2=I,%2,
ELSE
*
*  GET THE RUNTIME PACKAGE
*  AND EXECUTE THE PROGRAM
*
*  PUT BREAKN STATEMENT HERE
*  FOR MANUAL DATA INPUT
*
%2SDRUN,%1
*
*  IF NOT USING THE FLEX I CMMD, THEN
*  PUT DATA HERE FOR AUTO INPUT
*
END
```

JCP COMMAND SUMMARY

GETTING THE SYSTEM STARTED:

JCP,<filespec>[+]['<parameter list>]

Execute <filespec> with an optional list of parameters and turn off the echo of JCP lines if a "+" is entered.

COMMAND SUMMARY:

COMMAND	DESCRIPTION	PAGE
*	Comment.	6
. <labelname>	Label line.	6
GOTO <labelname>	Branch to <labelname> and continue execution.	7
BREAK	Suspend processing.	8
BREAKN	Suspend processing after the next line.	8
CONT	Continue processing.	8
IFSET <command>	Execute <command> if the condition code is set.	9
IFCLR <command>	Execute <command> if the condition code is clear.	9
SET	Set the condition code.	10
CLR	Clear the condition code.	10
%n=<string>	Replace the specified parameter with <string>.	10
IF %n=<string> . ELSE	If the specified parameter equals <string>, program execution continues with the next line; otherwise, branch to the next ELSE.	11
IFN %n=<string> . ELSE	If the specified parameter is not equal to <string>, program execution continues with the next line; otherwise, branch to the next ELSE.	11
CALL <labelname>	Branch to <labelname> and continue execution until a RETURN is encountered, then continue execution with the line following the last CALL.	12
RETURN	Return to the line following the last CALL.	12
ONERROR <command>	Execute <command> in the event of an error.	12
END	End procedure.	13
EON	Turn on the echo of JCP lines.	14
EOFF	Turn off the echo of JCP lines.	14
+<string>	Deliver <string> to FLEX for execution, then continue executing with the line following +<string>.	15

EDITOR COMMAND SUMMARY

GETTING THE EDITOR STARTED:

JCP[,<filespec 1>]

Call the editor and if <filespec 1> is entered and does not exist as a disk file, a new file will be created as specified by <filespec 1>.

COMMAND SUMMARY:

COMMAND	DESCRIPTION
P	Print the current line.
P!	Print from the current line to the last line and make the last line the current line.
N or (CR)	Make the next line the current line and print it.
-	Make the previous line the current line and print it.
T	Make the first line the current line and print it.
N!	Make the last line the current line and print it.
D	Delete the current line, make the next line the current line and print it.
=<string>	Replace the current line with <string> and print it.
I	Leave the command mode and begin line input after the current line.
#	Leave the insert mode; return to the command mode; make the last inserted line the current line and print it.

EXITING THE EDITOR:

FLEX

Write and close <filespec 1> if it was specified when the editor was called. Then return control to FLEX.

RUN or JCP[,][<filespec 2>][+]['<parameter list>]

Write and close <filespec 1> if it was specified when the editor was called. If <filespec 2> is not entered, execute the text created with the editor as a procedure file with an optional list of parameters. If <filespec 2> is entered, it will be used as the procedure file. The echo of JCP lines will default to off if the "+" is entered.