JOB CONTROL LANGUAGE

FOR LDOS 5.3 TRSDOS 6.1 6.2 LS-DOS 6.3

PRESENTED BY

Christopher Fara

JCL by Chris

Here finally is a common-sense presentation of the "Job Control Language" for Mod-III LDOS and Mod-4 TRSDOS/LS-DOS. The "official" documentation is full of intimidating terms like "macro" or "compiled language", and would make us believe that JCL must be something awfully "advanced". It is not. Once you organize it and get rid of the "powerful" words, JCL turns out to be rather simple. Read on, try out our examples, make up your own, and you'll soon agree: it's easy, often quite useful, always fun.

Printed in the United States of America

9 8 7 6 5 4 3 2 1

Published and Distributed by

Computer News 80

P. O. Box 680
Casper, Wyoming 82602-0680

JCL OUTLINE

Job Control Language, JCL, is a kind of simple "programming language for non-programmers", available on Mod-III (and I) under LDOS, and on Mod-4 under TRSDOS and LS-DOS. Compared with BASIC's 120+ commands, JCL is very simple indeed. Its vocabulary consists of some 20 instructions which can be used to enhance and modify the performance of the DOS command DO.

JCL FILES

The basic idea of DO is this. First, we create files which contain a series of lines such as DOS commands. Those files usually have the extension /JCL and are therefore commonly called "JCL-files". When we later DO a JCL-file, the "prefabricated" commands are fetched from the file, one line at a time, and processed just as if they were typed "live" from the keyboard. The kind and sequence of input lines must be carefully designed to match the sequence expected by any given application, but routine jobs can be nicely automated this way.

JCL INSTRUCTIONS

In addition to DOS commands and other inputs, JCL instructions can be inserted in a JCL-file to:

(1) Execute some action, such as display informative messages, pause, generate audible alert tones, etc. These are called EXECUTABLE instructions.

(2) Select from "raw" JCL-files only certain specific lines, modify those lines, and so on. These are called COMPILATION instructions, because they "compile", ie. put together an executable sequence of lines from a larger collection in the original "raw" files.

CONFUSING TERMS

"Macro" For some obscure reason the JCL instructions are called "macros" in the "official" documentation. But in general computer usage "macro" (from Greek meaning "long") is a feature which expands some short input (eg. a single keystroke) into a longer input text (eg. an entire phrase). The JCL "macros" do nothing that fancy. Each JCL instruction merely performs some specific simple action and that's all.

"Token" Another curious word, but it really means a sort of simple "variable" used by JCL in a similar way variables are used in BASIC programs.

"Compilation" Normally in computer terminology to "compile" means to use a "compiler" to translate a program written in a "high level" language such as BASIC or FORTRAN, into a /CMD-type machine program executable from DOS. The JCL "compilation" has nothing to do with that. It simply moves some lines from the original JCL-file to another JCL-file called SYSTEM/JCL.

DESIGNING JCL

Practically any DOS or BASIC command can be included in a JCL-file. In fact any input to any program (even /CMD-type machine program) can be "prefabricated" this way, as long as it is predictable. The only other restriction is this: it must be the kind of input which is 'entered', ie. some text typed and "terminated" by pressing the 'enter' key. Single key inputs (as in the BASIC command INKEY$) won't work from JCL-files.

To design a JCL-file for a procedure with numerous commands and inputs, it is best to first run it "live" and carefully note the sequence and spelling of all routine inputs, up to the first "non-routine" input. Then type the routine inputs into a JCL-file.

DOS COMMANDS

With only a few exceptions, any DOS command can be included in a JCL-file. Suppose each morning, after power-up, you want to:
. refresh memory and see a list of DOS commands
. make sure BASIC is on drive :0
. turn on the clock display on the screen
A simple 3-line JCL-file would do this job for you:

```
  LIB
  DIR BASIC:0 (INV)
  TIME (CLOCK=ON)
```

Commands which cannot be included in the JCL-file:

```
  BUILD                 DEBUG
  RESET                 SYSGEN
  SYSTEM (SYSTEM=)      SYSTEM (SYSGEN)
```

Also not allowed are any commands which
. require swapping of disks with the (X) parameter
. require pressing of a single key to continue
. include QUERY=ON (to prevent query specify Q=N in such commands as CONV, FORMAT, PURGE, etc).

BASIC COMMANDS

All BASIC commands, as well as responses to INPUT and LINE INPUT (but not single key responses to INKEY$) can be passed from JCL-files. For example

```
  BASIC
  RUN "PROGRAM/BAS"
  CHRIS FARA
  //STOP
```

enters BASIC and runs a program which asks for user's name as the first INPUT. The JCL instruction //STOP is required to leave control with the BASIC program and to prevent premature exit to DOS.

MACHINE PROGRAMS

Inputs requested by a /CMD-type machine program via Mod-3 CALL 64 ($KBLINE) or Mod-4 SuperVisory Call #9 (@KEYIN) are accepted from JCL-files. As in BASIC, such files must end with the //STOP instruction to prevent premature exit to DOS.

BUILDING JCL

The JCL-files are plain ASCII files, ie. they contain only human-readable characters and such "control" characters as "tab" or "carriage return". Each line in a JCL-file may have up to 63 characters (Mod-3) or up to 79 characters (Mod-4), and is terminated by "carriage return" ASCII code decimal 13. Any word processor capable of outputting text in pure ASCII format can be used to create and edit JCL-files (eg. use LDOS/LSDOS editor TED, or from Scripsit save such files with A-option). Or use BUILD command.

BUILD COMMAND

The DOS command BUILD is a sort of "poor man's word processor". It has a couple of limitations:

(a) Once a line is typed and 'entered' it cannot be edited. To correct a mistake or change something, we must re-BUILD the entire file "from scratch".

(b) To BUILD a file we must make sure it does not yet exist on one of the disk drives. If it does exist then BUILD aborts. In this case we must KILL (Mod-3) or REMOVE (Mod-4) that file, or pick a different name for our file and try again.

But the advantage of BUILD is that it's a "library command" always available at DOS READY prompt.

BUILD PROCEDURE

Suppose we want to BUILD a file called INFO/JCL. Verify that it does not exist yet and from DOS type

```
BUILD INFO 'enter'
```

The message "Building INFO/JCL" appears. Notice that if we don't specify extension then /JCL is automatically added to file name. Next type, for example

```
LIB 'enter'
DIR BASIC:0 (INV) 'enter'
TIME (CLOCK=ON) 'enter'
'break'
```

Pressing 'break' as the first key at the beginning of a line ends the BUILD process and returns to DOS (in Mod-4 we can also press 'control shift @' instead of 'break').

To execute all 3 commands we now only need to type

```
DO INFO 'enter'
```

Notice that the DO command automatically assumes that INFO has extension /JCL, and we don't need to type it. Had we "built" our INFO with an extension other than the "default" /JCL, then it would have to be typed in the DO command. However, extensions other than /JCL are normally not recommended.

DOING JCL

Our example from previous page
  DO INFO
is the simplest form of DO and does 2 things:

COMPILATION

First it selects ("compiles") executable lines from the original JCL-file and puts them into another file which is always called SYSTEM/JCL. If that file already exists then it is overwritten by the new information, otherwise it is created.

EXECUTION

After this "compilation phase" is completed, the "execution phase" begins: commands stored in SYSTEM/JCL are executed one at a time.

SYNTAX OF "DO"

The full syntax of the DO command has more options:

```
  DO [ control ] file [ (parameters,...) ] [;]
```

The 'file' normally is just the name: extension /JCL is assumed. Square brackets are never typed, but only indicate optional items.

control — A single character which must be separated by one space from DO. If this character is omitted, then both "compilation" and "execution" are processed as in our DO INFO example above. The 'control' can be:

= — The "equals" sign means: don't make SYSTEM/JCL, just execute what's in 'file'. In our INFO example we could skip the "compilation phase" and enter
  DO = INFO
This control option can only be used if 'file' does not contain any JCL "compilation" instructions.

$ — The dollar symbol does the opposite of '='. For example DO $ INFO only compiles SYSTEM/JCL but does not execute it. It is useful when we wish to LIST a complex SYSTEM/JCL before executing to be sure it is "compiled" correctly.

* — Asterisk means: execute SYSTEM/JCL. In this case we never specify 'file' or 'parameters'. Just enter
  DO *
But if SYSTEM/JCL has not been compiled yet then error "file not in directory" is generated.

parameters — Optional parameters, enclosed in round brackets, can influence the "compilation" of SYSTEM/JCL. See COMPILING JCL.

semicolon ; — The DO command with many parameters could exceed screen width. In that case type as many parameters as fit on the first line, close the round bracket, type semicolon and 'enter'. A question mark appears on the next line: open round bracket, type remaining parameters, close bracket and 'enter'.

COMPILING JCL

As noted before, JCL "compilation" does not translate the original "raw" ASCII file into some fancy "machine program", but merely moves lines from that file into another ASCII file called SYSTEM/JCL. So why bother?

MULTI-PURPOSE FILES

The main advantage of "compilation" is that from one "raw" file we can "compile" and then execute SYSTEM/JCL with varying contents to suit various situations. This not only saves re-typing work, but also conserves disk space. Under Mod-3 LDOS and Mod-4 TRSDOS/LS-DOS the shortest possible disk file uses 1.5K of disk space (one "granule"). That's about 1500 characters, enough for some 50-75 lines of a typical JCL file! Even a complex multi-purpose file can easily fit in one 1.5K file and replace several single-purpose files.

PARAMETERS

The manipulation of the contents of SYSTEM/JCL is done by specifying variables and/or labels in the parameter field of the DO command, in order to:
. pass variable values to JCL file, and/or
. select blocks of lines (procedures) for execution.

PASSING VALUES

Any alpha-numeric "string" of up to 32 characters can be passed to a variable ("token") specified in the JCL file, by assigning the string to the "token" in the parameter field of DO. For example

```
  DO FILENAME (P=BASIC)
```

would replace P by BASIC wherever the "token" P occurs in the "raw" file. Turn page to JCL TOKENS for more details about this feature.

SELECTING BLOCKS

Entire sequences of lines in the JCL-file can be designed to perform various procedures. These blocks of lines can be selected for execution by any of the following techniques:
. conditional blocks
. included blocks (from other JCL files)
. labeled blocks
. numbered blocks
The block selection techniques are described on two pages following the discussion of JCL TOKENS.

SAVING SYSTEM/JCL

We can compile and then save SYSTEM/JCL under some different name, for example like this

```
  DO $ FILE
  COPY SYSTEM/JCL SOMENAME
```

and later get any such pre-compiled file back

```
  COPY SOMENAME/JCL SYSTEM
```

This would seem to defeat one of the advantages of compilation noted above (conserving disk space by compiling SYSTEM/JCL as needed from one "raw" JCL-file) but may be useful if we frequently switch between various jobs and want to skip re-compilation, especially when it's a lengthy one.

JCL TOKENS

Like in BASIC, variables can be used in JCL files, except they are called "tokens" (another of those confusing terms). A "token" can have up to 8 letters and/or digits in any combination. Tokens are used

. in the //IF instruction to test if they are "true"
. to replace a token by a "string" of characters.

TRUE TOKENS

A token is "true" if it appears in the parameter field of the DO command. For example

```
  DO FILENAME (1,X2,A=BASIC)
```

would make the tokens 1, X2 and A all "true". Notice that, unlike BASIC variables, a JCL token can start with a digit and can even be just one single digit.

Manipulating the truth . . .

A token can be also made "true" within JCL-file. An instruction such as

```
  //SET XYZ
```

would make the token XYZ "true" even if it wasn't mentioned in the DO command. On the other hand

```
  //RESET XYZ
```

would make this token "not true". Even if we specify it in the DO command, it will be treated as if we didn't specify it!

Logical operators . . .

A "true" token can be also changed into "not true" by the "logical operator" NOT, represented by a minus sign. For example

NOT

```
  //IF -XYZ
```

means "if token XYZ is NOT true then compile next line".

The "logical AND" is represented by "ampersand". For example, a line like this

AND

```
  //IF A&B
```

means "if both A AND B are true then compile next line". If one of the ANDed tokes is not true, then the combination is also "not true".

The plus sign means "logical OR":

OR

```
  //IF A+B+C+D
```

Here the combination is "true" if at least one of the tokens is "true" (eg. A, B and C are "not true", but D is "true", etc).

Note: don't type spaces between tokens and operators because it may cause "illegal line format" error.

Muddling the truth . . .

Logical operators can be "mixed and matched" to express all kinds of logic. For example

```
  //IF A+-B&-C
```

means "if A is true OR B is NOT true AND C is NOT true, then go ahead, compile next line", etc. While such combinations are "legal", it's quite easy to get things confused this way. Don't overdo it.

See also JCL BLOCKS later in this section.

REPLACING TOKENS

If we enter a command such as

```
 DO FILENAME (F=BAS)
```

the "string" BAS will be assigned to the token F. In the JCL-file we might have a line like this

#token#

```
 DIR /#F#:0
```

A token between two "pound" symbols means: replace this by the string assigned to the token. In the resulting "compiled" SYSTEM/JCL we will find

```
 DIR /BAS:0
```

On another occasion we might

```
 DO FILENAME (F=CMD)
```

which would produce

```
 DIR /CMD:0
```

in the compiled SYSTEM/JCL. But DO FILENAME without parameters would "compile nonsense". The result would be DIR /#F#:0 because no replacement is available. So be careful. Of course such simple "one-liners" wouldn't justify bothering with BUILD and DO, and would most likely be part of a longer procedure. But the general idea how "tokens" can be used to produce various results from the same JCL file should be evident from the above examples.

In-file assignments . . .

A string can be also assigned to a "token" within JCL-file (instead of being passed from the DO command). Consider this JCL fragment:

```
 //IF -N
 //ASSIGN N=DATADISK
 //END
 FORMAT :1 (Q=N,ABS,NAME="#N#")
```

If we compile this without parameters

```
 DO FILENAME
```

then //IF -N (if token N is NOT "true") causes the compilation of the next line which assigns a default disk name (to prevent nonsense as in the previous example above). In the compiled SYSTEM/JCL we'll find

```
 FORMAT :1 (Q=N,ABS,NAME="DATADISK")
```

But if that file is compiled with a command such as

```
 DO FILENAME (N=DISK1234)
```

then the token specified in the parameter field is "true", the //IF test fails, //ASSIGN is skipped, the string DISK1234 is passed to the file and replaces the token N. The result will be

```
 FORMAT :1 (Q=N,ABS,NAME="DISK1234")
```

Quoted strings . . .

Sometimes double quotes must be used in //ASSIGN, if the string contains characters which could confuse the DO command. For example:

```
 //ASSIGN F="(INV)"
 DIR #F#
```

Here the idea is to produce DIR (INV) in the compiled SYSTEM/JCL. But without the quotes (INV) would be mis-interpreted by DO as a "programmers comment" (see JCL MESSAGES later in this section) and nothing would be assigned to the token F.

JCL BLOCKS

One "raw" JCL-file can contain several "blocks" (sequences) of lines designed to perform various job "procedures". When a procedure is needed, a block of lines can be selected for execution by one of the following methods.

CONDITIONAL BLOCKS

Conditional blocks start with the JCL instruction

```
//IF token
```

As mentioned earlier, "token" is a kind of variable. If the token in the //IF is "true" then lines following this instruction are compiled, up to

```
//ELSE or //END
```

whichever comes first. If the token is not true and there is an //ELSE then lines following the //ELSE, up to //END are compiled. The //ELSE instruction is optional, but for every //IF there must be always a matching //END. For example this fragment

```
//IF X
DIR
//ELSE
FREE
//END
```

would display directory if token X is "true", otherwise it would display free disk space. Of course in practical JCL-files not just one but several command lines could be typed in each block between //IF and //ELSE, or between //ELSE and //END.

A token is "true" if it is specified in the parameter field of the DO command, or is //SET or //ASSIGNed in the JCL-file. Logical operators NOT, AND, OR (represented by - & + symbols) can be used to modify the "truth" of the //IF test. For details and examples see JCL TOKENS earlier in this section, and the //IF instruction in the reference section.

INCLUDED BLOCKS

The instruction

```
//INCLUDE filespec
```

temporarily suspends the processing of the current JCL-file and fetches lines from another JCL-file. Its lines are compiled, and when the "included" file is done, then compilation of the main file resumes. For example a fragment like this

```
//IF X
//INCLUDE NEXTFILE
//END
```

would test if the token X is "true" and if so, it would fetch some lines from NEXTFILE/JCL. But if the token X were "not true" then the lines from NEXTFILE would not be included.

Included files may in turn //INCLUDE other files, and so on, up to 5 (Mod-4) or 10 (Mod-3) "nested" levels! The only other restriction is that //INCLUDE must never be the last line in any JCL-file.

LABELED BLOCKS

Labeled blocks start with a label

```
  @ANYLABEL
```

which must begin with the @-symbol and may consist of up to 8 letters and/or numbers in any combination. When a JCL-file with labels is compiled with a command such as

```
  DO FILENAME (@LABEL)
```

and the label specified in the parameter field matches one of the labels in the "raw" file, then lines following that label are compiled (ie. moved to SYSTEM/JCL), up to
. another label
. end of file
whichever comes first. If the label is not specified then lines from the start of file up to any first label are compiled. For example

```
  . Specify a label
  @FIRST
  . Compile first block only
  @SECOND
  . Compile second block only
  . and so on
```

In this example DO FILE (@FIRST) would display the message "Compile first block only", but DO FILE without parameters would display "Specify a label".

Note that a label should never be the first line in any JCL-file. Any number of different labels can be included in a JCL-file, but only one can be selected for processing in the DO command. Labels and tokens can be combined in the parameter field.

NUMBERED BLOCKS

Numbered blocks are similar to labeled blocks, but start with different markers

```
  //number
```

which can be //0, //1, etc, up to //9. Another difference is that a numbered block is not "compiled" but simply selected for execution by pressing a matching digit key in response to the instruction

```
  //KEYIN
```

which must appear ahead of the numbered blocks. If the pressed digit matches one of the blocks then lines following the block marker are executed up to
. an //ABORT, //EXIT or //STOP instruction
. next numbered block marker
. end of file
whichever comes first. A "default" block marker /// should be provided in such files: if the pressed key does not match any numbered block, then lines following the /// are executed.

Since the numbered blocks are not compiled but simply executed when a matching digit key is pressed, they can be also used in non-compiled files (ie. files "done" with the '=' option). All other types of blocks require compilation.

JCL MESSAGES

Messages and comments in JCL-files make the JCL process more "user friendly". Also, even the author of a JCL-file might forget after a while what that particular file was meant to accomplish. Comments help refresh our memory.

EXECUTABLE MESSAGES

Executable messages are displayed during "execution phase" of the DO processing. Two instructions simply display a message

```
  .message (must start with dot, steady display)
  //FLASH message (flashing display)
```

Messages can be also optionally displayed by

```
  //INPUT
  //KEYIN
  //PAUSE
```

to "prompt" the operator for appropriate action (press a key, insert diskette, etc).

In addition, an audible tone or series of various tones (which is a sort of "message" for the ears rather than eyes) can be produced by the instruction

```
  //ALERT
```

Mod-4 has a built-in tone generator. Mod-3 needs an amplifier connected to cassette jack.

Display of executable messages can be enhanced by inserting ASCII "control characters" in a special hex-coded format. For example %09 inserts a "tab", %0A inserts a "line feed", etc.

COMPILATION MESSAGES

Normally nothing appears on the screen during the "compilation phase". If we want to be informed what is happening (particularly during a compilation of a lengthy and complex JCL-file) then we can insert

```
  //.message (double slash and dot)
```

which is similar to the "executable" dot-message, but displays the message only during the compilation phase and is never moved to SYSTEM/JCL. The display of "compilation messages" cannot be enhanced with "control characters".

PROGRAMMER'S COMMENTS

Any "compilation instruction" can have a comment added at the end of the line. It must be enclosed in round brackets. For example in a line like this

```
  //IF N      (TEST IF DISK NAME IS SPECIFIED)
```

the text in brackets is a comment. This kind of comment is never moved to SYSTEM/JCL (because "compilation instructions" only control compilation process and are not moved to SYSTEM/JCL), and it is not even displayed during compilation. But it can be viewed when the JCL-file is LISTed on the screen or printer. The "programmers comments" can be only added to "compilation" instructions, never to any other lines in a JCL-file.

JCL SYMBOLS

In addition to the JCL instructions, certain symbols have special meaning in JCL-files:

"	Double quote: must enclose "string" in //ASSIGN if the "string" contains characters which could be mis-interpreted by the DO command. See JCL TOKENS and //ASSIGN.
#	Pound symbol: if a token is enclosed within pound symbols then in the "compiled" SYSTEM/JCL it will be replaced by a "string" specified in the parameter field of the DO command or //ASSIGNed to it in the JCL-file. See JCL TOKENS and //ASSIGN.
%	Percentage sign: means that two characters following it represent a hex number. Any ASCII character can be created this way, but only if file is "compiled". For example a line like this `%44%49%52` representing ASCII codes for D, I, R, would produce `DIR` in the compiled SYSTEM/JCL. Normally, however, % is used to insert "control characters" into JCL-files, such as 'clear screen', 'tab', 'break', etc. See also .message ("dot-message") in the reference section.
&	Ampersand: means "logical AND". Used in the //IF instruction to check if two or more tokes are all "true". See JCL TOKENS and //IF.
()	Round brackets: enclose "programmer's comment" in "compilation instructions". See JCL MESSAGES.
+	Plus: means "logical OR". Used in //IF instruction to test if at least one token is "true". See JCL TOKENS and //IF.
-	Minus: means "logical unary NOT". Used in the //IF instruction to cause compilation if token is "not true" (opposite of normal "if true"). See JCL TOKENS and //IF.
.	Period (dot): must be the first character of a displayable message. See JCL MESSAGES.
//	Double slash: except for .message and @label all JCL instructions must begin with double slash.
=	Equals symbol: used in //ASSIGN instruction to assign a "string" to a token. See //ASSIGN.
@	At-symbol: must be the first character of a label, both in the JCL-file and in the parameter field of the DO command, if a label is specified. See JCL BLOCKS and @LABEL.

JCL ERRORS

Like any programming language, JCL is picky and insists on exact spelling. All required dots, slashes, etc, must be typed just so. Errors usually abort JCL processing, with or without explanatory message. Some typical error messages and probable causes are:

"File already exists"	Attempt to BUILD a file under an existing name.
"File not in directory"	You enter DO * but SYSTEM/JCL has not been "compiled" yet.
"Can't create SYSTEM/JCL"	System diskette is write-protected.
"End of file / out of range"	//INCLUDE instruction in the last line of JCL-file.
"Too many INCLUDEs"	Too many "nested" levels of //INCLUDE.
"Line too long"	More than 63 (Mod-3) or 79 (Mod-4) characters.
"Illegal line format"	Usually caused by syntax errors, for example spaces between tokens and "logical" operators (eg. A+-B&-C is legal, but A + -B & -C usually generates error).
"String too long"	String longer than 32 characters assigned to a token.
"Multiple definition"	Two or more values assigned to the same token.
"Too many labels"	More than one label specified in the DO command.
"Procedure not found"	Label specified in the DO command does not match any label in the JCL-file.

Other mistakes may produce all sorts of curious errors, depending on the situation:
. //IF without matching //END.
. Some types of //ASSIGNED strings without quotes.
. //FLASH or //KEYIN messages "enhanced" with %.
. JCL instruction typed in first line of JCL-file (to prevent this error, always start your JCL-files with a "dot-message").

TRIAL COMPILATION

The best way to avoid serious disasters, especially with complex JCL-files, is to first "compile" with the DO $ option. Then LIST and carefully review the resulting SYSTEM/JCL to make sure it contains the desired lines. When everything looks good, then simply DO * to execute it.

K.I.S.S.

You know, "Keep It Simple, S... ", avoid confusing logic such as too many "nested" //IFs, complex "token expressions" with many NOTs ANDs ORs, multiple //SET //RESET //ASSIGN, etc. Amazing results can be achieved with such "advanced techniques", but equally amazing mess is also quite possible.

JCL REFERENCE

In this section all JCL-instructions are summarized in their "logical" order on the next 2 pages, followed by detailed descriptions in alphabetical order.

REFERENCE FORMAT

Each instruction in the alphabetical listing has a separate bold heading, followed by a short title and the word
  EXECUTE
which means that this is an "executable" instruction and may be used in JCL-files "done" with the DO = option or in "compiled" files. The word
  COMPILE
indicates "compilation" instructions which are used to manipulate the compilation process and are never moved to the resulting SYSTEM/JCL. They cannot be used in files executed with the DO = option.

[ ] Square brackets indicate optional items which may be omitted at user's discretion. These brackets are not typed in the JCL-file.

UPPER / LOWER CASE

Instructions may be typed in upper or lower case, or even mixed, it doesn't make any difference. For example //ALERT is the same as //alert or //Alert. However, we use here upper case to indicate words which must be typed and spelled exactly as shown, and lower case to indicate items such as messages to be composed by the user.

EXAMPLES

Each instruction is illustrated by simple examples written so the user may type them as-is and see how things work. The examples are not intended to be terribly practical, but should encourage the reader to experiment. As with any programming language, the best way to learn is to play with it a bit, see what happens. Type the examples into a JCL-file, then "compile" with DO $ option, list SYSTEM/JCL to see what you've got, and if it looks good, DO * to execute SYSTEM/JCL and see if it does what it was supposed to do.

REMEMBER

- Each instruction must be entered on a separate line.
- Except for the "dot-message" (.message) a JCL instruction can't be the first line of any JCL-file. But DOS or BASIC commands and other inputs are legal in the first line.
- The //INCLUDE instruction can't be the last line.
- The 'break' key can be pressed any time to abort processing of JCL-files.

SUMMARY OF INSTRUCTIONS

"Executable" instructions perform indicated action, whether "compiled" or not.

"Compilation" instructions manipulate the compilation process and are never moved to the resulting file SYSTEM/JCL. They cannot be used with DO = option.

Each instruction must be entered on a separate line.

J C L e x e c u t a b l e i n s t r u c t i o n s

. message	. Display 'message' and continue (must start with dot).
//FLASH count message	. Flash 'message' specified 'count' of times (0 to 255).
//ALERT pitch,silence,....	. Produces a tone (Mod-3 only if amplifier connected to cassette jack). 'Pitch' can be 0 (high pitch) to 7 (low) always followed by a matching 0 (short silence) up to 7 (long silence). If the sequence is enclosed in brackets such as //ALERT (0,0,1,0,2,0,3,0) then it sounds indefinitely, until 'enter' is pressed to continue, or 'break' to return to DOS.
//PAUSE message	. Display optional 'message', wait for 'enter' to continue, or 'break' to return to DOS.
//DELAY duration	. Just pause a few seconds, no message. 'Duration' is a number from 1 (0.1 sec) through 256 (25.6 seconds).
//SLEEP hh:mm:ss	. (Mod-4 only) Suspend JCL processing for a specified period of hours, minutes, seconds.
//WAIT hh:mm:ss	. Pause until hh:mm:ss matches system time.
//INPUT message	. Display optional 'message', wait for input from keyboard terminated by 'enter', accept this input as if coming from JCL file, and continue.
//KEYIN message	. Display optional message, wait until one of the keys 0-9 is pressed, then proceed to execute selected block of lines in the JCL file. Blocks start with //number (for example //0 or //3 etc) and end with another //number, or triple slash /// (just 3 slashes in one line) which means: if none of the blocks match the number keyed-in, then proceed to line following ///.
//EXIT	. Job done, return to DOS (not required if end of file).
//STOP	. Job done, but stay where you are (eg. in BASIC).
//ABORT	. Job not done, but return to DOS (eg. error detected).

@label	. If matching '@label' is specified in DO command then "compilation" selects only lines following the '@label' in the "raw" JCL file, up to any next '@label' or to end of file.
//IF token	. Compile lines following this instruction, up to //ELSE or //END, but only if 'token' is "true".
//IF -token	. Compile if 'token' is NOT "true" (opposite of above).
//IF token1 & token2	. Compile if 'token1' AND 'token2' are "true".
//IF token1 + token2	. Compile if 'token1' OR 'token2' is "true" (or both).
//ELSE	. Compile from //ELSE to //END if token is not "true".
//QUIT	. Should be placed after //ELSE to abort compilation if token is not "true" and no alternate action is desired.
//END	. Must terminate each //IF sequence. Sequences may be nested. Each //IF must have a matching //END.
//SET token	. Makes 'token' "true" even if not specified in DO.
//RESET token	. Makes 'token' "not true" even if specified in DO.
//ASSIGN token="string"	. Replaces 'token' by a 'string' of up to 32 characters, and makes 'token' "true". Quotes may be omitted if string does not contain characters which could confuse DO.
#token#	. Replaces 'token' between two # symbols by a string specified in DO or //ASSIGNed to it.
%hh	. Percentage symbol followed by a 2-digit hex code inserts control characters in messages, for example .%09 This comment is displayed at first tab stop //PAUSE %0A%0A This is displayed two lines down %1F .Clear screen before displaying this comment
//INCLUDE filespec	. Another JCL file is fetched from disk and its lines are "compiled". When done, compilation of the main file continues. Included files can //INCLUDE other files, and so on, but //INCLUDE can't be the last line in any file.
//.message	. Similar to "executable" message but displayed only during compilation. Never goes into SYSTEM/JCL.
//END (comment)	. Comments in brackets may be added to compilation instructions. They are never displayed but may be seen when file is LISTed.

. message

The "dot message": message display EXECUTE

. [message]

Any line in a JCL-file beginning with a "dot" is simply displayed as-is. If 'message' text is omitted, then a blank line is "displayed".

I M P O R T A N T . . .

No other JCL instruction, except the "dot message" is allowed in the first line of any JCL-file. Therefore it is a good practice to always have a "dot message" in the first line: a note what the file does, date, etc.

Enhanced messages . . . %

Percentage symbol followed by 2-digit hex code can be used to insert ASCII "control codes", such as

```
  %09  tab
  %0A  line feed
  %1F  clear screen
```

The "clear screen" code must be placed ahead of the "dot". The "tab" and "line feed" must be placed after the "dot" or within the message text, as desired. Other codes are not recommended (may cause errors).

#token#

A token between two "pound" signs can be replaced by a string assigned to it in the DO command or by the //ASSIGN instruction. Variable messages can be thus produced from standard message "templates". Note that a message such as

```
 . Insert disk #1
```

causes error, because the DO command "thinks" that '1' is a "token" (but "disk ##1" would be OK).

The "enhancing" works only if the file is "compiled".

See also . . .

JCL Outline: MESSAGES, and //. //FLASH //PAUSE

```
.                        . the "dot" alone may be placed in the first line
.My first JCL file       . or "dot message",
%1F.Clean message        . or "clear screen" (ahead of "dot") plus message,
//ALERT 1,2,1,2,1,2      . to make sure that a JCL instruction is not first
.%0A%0ATwo lines down    . "line feeds" display message one or more lines down
```

//. message

Message display COMPILE

//. message

Double slash, dot, message: like the executable "dot-message" above, but displayed only during "compilation". It is never moved to SYSTEM/JCL and never displayed in "execute" phase. Can be used to inform the programmer what's happening, particularly during a lengthy compilation. This "compilation message" cannot be "enhanced" with % or # symbols.

///

Default block marker — EXECUTE

///

If none of the keys pressed in response to the //KEYIN instruction matches a numbered block in the file (see //number below) then DO searches for a block marked by this "default" /// and if found, the lines following it are executed.

Always provide /// . . .

If numbered blocks are used then /// should be also provided, even if it's simply the last line in file. If /// is not provided and the key pressed in response to //KEYIN does not match any numbered block then the computer may hang up.

See also . . .

//number //KEYIN

//number

Numbered block marker — EXECUTE

//number

Marks the beginning of a block of lines. The 'number' must be single digit 0 through 9. If a numeric key pressed in response to //KEYIN matches the block then lines following the marker are executed up to
. //ABORT //EXIT or //STOP
. next numbered block marker
. end of file
whichever comes first. However, if the "default" /// marker is found before any of the above, then execution continues with lines following /// (because the /// marker is a "match" for "any key"). This may be intentional, if we wish those lines executed in any event, regardless of the //KEYIN response.

See also . . .

/// //KEYIN

. Numbered blocks example	. comment line
//KEYIN Press 0 or 1	. wait for key
//0	. if 0 pressed
DIR :0	. then display directory of drive :0
//EXIT	. and "job done"
//1	. if 1 pressed
DIR :1	. then display directory of :1
///	. if any key pressed
LIB	. then display list of DOS "library commands"

Note there is //EXIT at the end of block 0, but not at the end of block 1. Therefore if we press '0' then directory is displayed and we're done. If we press '1' then directory is displayed and then also LIB is executed. If we press any other key (even 'break') then only the LIB command is executed.

//ABORT

Terminate if error EXECUTE

//ABORT

Display message "Job aborted" and return control to DOS. Used to terminate processing if error is found.

See also . . .

//EXIT //QUIT //STOP

```
. Error example          . comment
//KEYIN Press 0 or 1     . wait for key
//0                      . if 0 pressed
DIR :0                   . then display directory of drive :0
//EXIT                   . and "job done"
//1                      . if 1 pressed
DIR :1                   . then display directory of :1
FREE                     . and free space on all disk drives
//EXIT                   . and "job done"
///                      . if any other key pressed
.Wrong key !             . then display explanation
//ABORT                  . and exit with the message "job aborted"
```

//ALERT

Generate sound EXECUTE

//ALERT pitch,silence,...

Produces a tone. 'Pitch' can be a number from 0 (high pitch) through 7 (low pitch), and must be always followed by a matching 'silence' number 0 (short silence) through 7 (long silence). Numbers higher than 7 are evaluated "modulo 8" (eg. 8 does the same as 0, 9 same as 1, 17 same as 1, 18 same as 2, etc). As many 'pitch,silence' pairs can be typed as fit on one line and can produce quite a music!

//ALERT (pitch,silence,...)

If the number pairs are enclosed in round brackets then the tune repeats indefinitely, until 'enter' is pressed to continue execution, or 'break' to abort.

Model III . . .

On Mod-III an amplifier must be connected to the AUX pin of the cassette jack (Mod-4 has a built-in sound generator).

```
. Put disk in drive 1, press ENTER (BREAK to quit)
//ALERT (7,0,6,1,5,2,4,3)
DIR :1
```

In this example alert sounds until we insert disk and press 'enter' to display directory or 'break' to abort.

//ASSIGN

Assign string to a token COMPILE

`//ASSIGN token = "string"`

Does the same thing within JCL-file as assigning a string to a token in the parameter field of the DO command. For example either the command

```
  DO FILE (A=/BAS)
```

or the instruction in the FILE/JCL

```
  //ASSIGN A=/BAS
```

would produce the same result. In either case the maximum length of 'string' must not exceed 32 characters.

Quotes around the string may be omitted if the string does not contain any characters which could confuse the DO command (see example below).

The //ASSIGN instruction also automatically makes the token "true", just as if it was specified in the DO command.

```
. Assign and substitute
//IF 1                        . check if token '1' is valid
//ASSIGN F = /BAS             . if so then assign the string /BAS to a new token F
//END                         . mandatory end of first block
//IF 2                        . check token '2'
//ASSIGN F = "(INV)"          . if valid then assign another string to token F
//END                         . end of second block
//IF 1+2                      . now test if at least one of the tokens is valid
DIR #F#                       . if so, then replace token F by the assigned string
//ELSE                        . otherwise
//FLASH 10 Specify token!     . flash advisory message 10 times
//QUIT                        . and abort compilation
//END (all done)              . "programmer's comment" may be added in brackets
```

Notice that unlike most symbols in DOS and BASIC (filenames, variables, etc) a JCL-token can start with a digit or can be just one digit. If you call this example file TEST/JCL then

```
  DO TEST (1) produces       DIR /BAS
  DO TEST (2) produces       DIR (INV)
  DO TEST without any token parameters aborts job.
```

Notice also that quotes around assigned strings are optional in most cases. They are only required if the string contains characters which could confuse DO. For example

```
  //ASSIGN F = (INV)
```

would not produce DIR (INV) because the round bracket would be interpreted by DO as "programmer's comment" and nothing would be assigned.

//DELAY

Short pause — EXECUTE

//DELAY duration	Just pause a few seconds, no message. The 'duration' is a number from 1 (about 0.1 second pause) through 256 (about 25.6 seconds).
See also . . .	//SLEEP //WAIT

```
DEVICE              . display device list
//DELAY 150         . pause about 15 secs to view the display
DIR (INV,SYS)       . then scroll-in the directory display
```

Notice in this example we don't have our usual "dot message" in the first line. Normally we recommend it to make sure a JCL instruction does not appear in the first line. But DOS or BASIC commands, or any other kinds of input are perfectly OK in the first line of any JCL-file.

//ELSE

Alternate conditional block — COMPILE

//ELSE	Can be used in //IF //ELSE //END sequence to mark an alternate block of lines in case the //IF test fails.

For details and examples see //IF

//END

End of conditional block(s) — COMPILE

//END	Must be used in //IF //ELSE //END sequences to mark the end of conditional block(s) of lines. For every //IF there must be a matching //END.

For details and examples see //IF

NOTES:

// EXIT

Terminate JCL processing EXECUTE

//EXIT

Display "Job done" and return control to DOS. This is required only if termination is desired at a point other than the last line of a JCL-file (end of file causes "implied" exit).

See also . . .

//ABORT //QUIT //STOP

```
. Exit example               . comment
//KEYIN Press 1 or 2         . wait for key
//1                          . if 1 pressed
DIR                          . then display directory
//EXIT                       . and "job done"
//2                          . if 2 pressed
FREE                         . then display free disk space
DEVICE                       . and the list of "devices"
//EXIT                       . and "job done"
///                          . if neither 1 nor 2 pressed
.Try again !                 . then display message
DO = TEST                    . and start over
```

If this file is called TEST/JCL then pressing 1 or 2 in response to //KEYIN executes some DOS commands and terminates with "job done". Note how we created an endless "loop" in the last line which restarts TEST until we press a correct key!

//FLASH

Flash a message EXECUTE

//FLASH [count] message

Flash the 'message' text specified 'count' of times and continue. The 'count' is a number 0-255. If the 'count' is omitted or 0 is specified then the message flashes 256 times. While the message is flashing, we may press 'enter' to continue execution of a next line, or 'break' to abort JCL processing. Note that unlike the steady "dot-message" (see .message) the flashing message can't be "enhanced" with % symbols.

```
. Flash example
//FLASH Put disk in drive 1, press ENTER (BREAK to quit)
FORMAT :1 (Q=N,ABS,NAME="MYDISK")
```

Here the 'count' is not specified so the message flashes up to 256 times, surely long enough to notice it and make sure the disk is inserted. However, if we fall asleep, then after 256 flashes the FORMAT will be attempted anyway, even if the drive is empty.

//IF

Conditional block start — COMPILE

```
//IF token
...
//ELSE
...
//END
```

If token is "true" (ie. specified in the DO command or //SET or //ASSIGNed in previous lines of the file) then lines following the //IF are compiled (moved) to SYSTEM/JCL, up to
 //ELSE or //END
whichever comes first. The //ELSE is optional. If provided and the //IF test fails, then lines from //ELSE to //END are compiled.

For every //IF there must be a matching //END.

Token expressions . . .

The "truth" of several tokens can be tested at once by using "logical operators":

//IF -token	-	minus means "if token NOT true then compile"...
//IF token1&token2	&	ampersand means "if token1 AND token2 are true..."
//IF token1+token2	+	plus means "if token1 OR token2 is true..."
//IF A+B&-C		

Logical operators can be combined to build more complex "truth expressions" (no spaces should be typed between tokens and operators). The operators are evaluated strictly from left to right. Brackets cannot be used to alter the order of evaluation.

Nested //IF . . .

The //IF //ELSE //END sequences may be "nested", but for every //IF there must be a properly matched //END. The general "nesting" scheme looks like this

```
 //IF token1
 . Compile first block of lines
 //IF token2
 . Compile second block
 //END (end of the "inner" test of token2)
 //END (end of the "outer" test of token1)
```

In this scheme the inner test is performed only if the outer test is "true". If the outer test fails then the inner test is not attempted at all.

Provide //QUIT . . .

The "compilation instruction" //QUIT should be provided after //ELSE in case the //IF test fails but no alternate procedure is provided in the JCL-file. The //QUIT aborts any further processing of DO and thus also prevents execution of SYSTEM/JCL which at this point might be useless (for example, the test fails because we have forgotten to assign a string to a token, but without that string the JCL procedure is likely to "bomb" the system).

See also . . .

JCL Outline: JCL BLOCKS

Continued on next page ...

```
. Simple //IF example      . comment, as usual to start the file
//IF D                     . is token D "true" (specified in DO command)?
//SET M                    . yes, so make token M also "true"
DEVICE                     . and execute DOS command
//END                      . end of first conditional block
//IF M                     . is token M "true" ?
MEMORY                     . yes, execute DOS command
//END                      . end of second conditional block
FREE                       . in any case display free disk space
```

When this file is executed with

```
DO FILE (D)
```

then the following SYSTEM/JCL is compiled

```
. Simple //IF example
DEVICE
MEMORY
FREE
```

because the "true" D caused M to be //SET "true". On the other hand

```
DO FILE (M)
```

creates

```
. Simple //IF example
MEMORY
FREE
```

because in this case only M is true. Finally

```
DO FILE
```

without any parameters creates

```
. Simple //IF example
FREE
```

because both D and M tests failed.

```
. Nested //IF example      . comment
//IF D                     . is token D "true" ?
DEVICE                     . yes, execute DOS command
//IF M                     . and see if token M is also "true" ?
MEMORY                     . yes, execute another DOS command
//END (inner block)        . note "programmers comment" in brackets
//ELSE                     . in case the "outer" test failed
//. Specify (D) or (D,M)   . display "compilation comment"
//QUIT                     . and abort DO processing
//END (outer block)        . this //END matches the first //IF
```

Here the following combinations are possible:

```
DO FILE (D,M)
```

compiles and executes

```
. Nested //IF example
DEVICE
MEMORY
```

because both tokens are "true". But

```
DO FILE (D)
```

compiles only the DEVICE command. Finally

```
DO FILE (M) or DO FILE
```

displays "Specify (D) or (D,M)" and aborts the DO processing. The "outer" test fails, and therefore the "inner" test is not even attempted, even if we specify the token M in the DO command.

//INCLUDE

Include lines from another file — COMPILE

//INCLUDE filespec

The "compilation" of the main file is temporarily suspended and some lines are fetched from another JCL-file. After that "included" file is "done", the processing of the main file resumes at the line immediately following //INCLUDE. The 'filespec' must be a name of a valid JCL file. The extension /JCL is assumed and we don't need to type it.

The //INCLUDE instruction cannot be the last line in any JCL-file.

Nesting . . .

The "included" file may in turn //INCLUDE another file, and so on, up to 5 (Mod-4) or 10 (Mod-3) "nested" levels.

```
. This is FIRST/JCL
. First block of lines
. This is SECOND/JCL
. Second block of lines
. Third block of lines
```

The SYSTEM/JCL on the left would be produced by the command DO FIRST from the 2 files listed below:

```
 . This is FIRST/JCL      . This is SECOND/JCL
 . First block of lines   . Second block of lines
 //INCLUDE SECOND
 . Third block of lines
```

Not a very practical example, but you get the idea.

//INPUT

Line input — EXECUTE

//INPUT [message]

Display optional 'message' and wait until something is typed and terminated by pressing 'enter'. This input is accepted and processed from the "live" keyboard as if it was coming from the JCL-file, and then the DO processing resumes. The input must be a valid command at the level being processed (DOS, BASIC, etc) and error-free, otherwise further processing of the JCL-file is aborted. Also, don't use //INPUT if //PAUSE or //ALERT are used in any preceding line (an 'enter' pressed in response to those instructions can "carry over" to //INPUT and abort processing). The 'message' should not be "enhanced" with % or # symbols (they cause all kinds of funny errors here).

```
. Input example
//INPUT Enter TIME hh:mm:ss
//WAIT 13:00:00
...
```

This fragment of a JCL-file could be used to enter the DOS command TIME to make sure the system time is set correctly for the //WAIT instruction.

//KEYIN

Single-key input EXECUTE

//KEYIN [message]

Display optional 'message' text and wait until a key is pressed. If one of the numeric keys 0-9 is pressed and it matches a "numbered block" in the JCL-file, then execute lines in that block. Otherwise execute lines in the "default" block (if provided).

The //KEYIN instruction must appear in the JCL-file ahead of the numbered blocks which it selects.

The 'message' should not be "enhanced" with the % symbols (all kinds of strange errors may result).

For details and examples see /// and //number earlier in this reference section.

//PAUSE

Pause and wait EXECUTE

//PAUSE [message]

Display optional message and wait until 'enter' is pressed to continue, or 'break' to abort further processing. The 'message' may be "enhanced" with the % symbols (see "dot-message" at the beginning of this reference section), but %1F (clear screen) does not always work with //PAUSE, so check it out first.

```
. Pause example
//PAUSE Put disk in drive 1, press ENTER (BREAK to quit)
FORMAT :1 (Q=N,ABS,NAME="DATADISK")
DO = TEST
```

If we call this file TEST/JCL then it could be used to format a bunch of "data disks" in drive 1, because this example is an "endless loop" (the last line starts over). Just keep inserting blank disks and press 'enter'. When all disks are formatted, press 'break' to return to DOS.

//QUIT

Abort compilation COMPILE

//QUIT

Should be used in //IF //ELSE //END sequences to abort compilation in case the //IF test fails and no alternate action is possible.

For details and examples see //IF.

//RESET

Make token "not true" COMPILE

//RESET token

This is inverse of //SET. Even if token is specified in the DO command, we force it to be "not true".

See also . . . //SET

```
. Example of RESET
//IF A                  . is token A "true" ?
DIR                     . yes, display directory
//RESET B               . and make B "not true"
//END                   . end of first block
//IF B                  . is token B "true" ?
FREE                    . yes, display free disk space
//END                   . end of second block
```

This scheme can be used to force an "either-or" type of compilation. Consider

DO FILE (A)

displays directory. Token B is not specified and therefore the second block is skipped.

DO FILE (B)

skips first block and compiles only the second part.

DO FILE (A,B)

displays directory and makes B "not true", thus making sure that the second block is skipped.

//SET

Make token "true" COMPILE

//SET token

Does the same thing within JCL-file as specifying 'token' in the DO-command. For example either the command DO FILE (A) or the instruction //SET A within the file would result in token A being "true".

See also . . . //RESET

```
. Example of SET
//IF A                  . is token A "true" ?
//SET B                 . yes, then make B also "true"
. First block           . and compile the first block of lines
//END                   . end of first block
//IF B                  . is token B "true" ?
. Second block          . yes, compile second block
//END                   . end of second block
```

Here we can DO FILE (A) or DO FILE (A,B) and in both cases both blocks will be compiled, because a "true" A forces B to be also set "true". This scheme can be used to make sure a needed procedure is compiled even if we forget to specify a token.

//SLEEP

Long pause (Model 4 only) EXECUTE

//SLEEP hh:mm:ss	Suspend execution for a specified length of time (hours, minutes, seconds) and then resume processing. Pressing 'break' interrupts the "sleep" and aborts JCL processing.
See also . . .	//DELAY //WAIT

```
...
//SLEEP 00:15:00
...
```

. just 15 minutes
. insert a few such lines at strategic points in your file as an excuse for coffee breaks or short naps.

//STOP

Terminate JCL processing EXECUTE

//STOP	Return control to the "live" keyboard but "stay where you are". Use //STOP to prevent premature exit from a BASIC program (or machine program). If we don't use //STOP and the end of JCL-file or //EXIT is reached when our program requests an input, then the program is immediately interrupted and control returns to DOS!
See also . . .	//ABORT //EXIT //QUIT

```
. Example of //STOP
BASIC GAME/BAS (F=1)
CHRIS, HEIDI, EVA
//STOP
```

. load BASIC and run GAME/BAS with one open file
. prefab input to a prompt "Input 3 players names"
. //STOP makes sure that any following INPUT request can be answered "live" from the keyboard.

//WAIT

Pause until specified time EXECUTE

//WAIT hh:mm:ss	Suspend execution until specified time matches the system time (expressed in standard 24-hour format, hours, minutes, seconds). Pressing 'break' interrupts the waiting and aborts JCL processing.
See also . . .	//DELAY //SLEEP

```
...
//WAIT 13:00:00
...
```

This fragment of a JCL-file would make sure nothing happens until you get back from lunch.

@LABEL

Labeled block marker — COMPILE

@label

The label must begin with the @-symbol and can have up to 8 letters and/or digits in any combination. Even a single digit will do, such as @1 (but don't confuse this with the "numbered" blocks). Labels are used to mark the beginnings of blocks of lines in a "raw" JCL-file. Each block can contain any number of lines. When a JCL-file with labeled blocks is "done" with a command such as

```
  DO FILE (@LABEL)
```

and the label specified in the parameter field matches one of the labels in the file, then those lines which follow the selected label are "compiled" (ie. moved) to SYSTEM/JCL, up to
. another label
. end of file
whichever comes first. Any number of different labels and procedures of any length can be included in the file, but only one label may be selected for compilation by the DO command.

If label is specified in the DO command, but does not match any label in the file, then error "procedure not found" occurs and compilation is aborted.

If label is not specified in the DO command, then only lines from the beginning of file up to any first label are compiled. For this reason a label should never be the first line in any JCL-file (start the file with a "dot-message" or a DOS or BASIC command).

See also . . . JCL Outline: JCL BLOCKS

```
. Labeled blocks
CLS                     . clear screen
TIME (CLOCK)            . turn on clock display
@B                      . label B
DIR /BAS:0 (A=N,P)      . if selected then print "short" directory of /BAS files
@C                      . label C
DIR /CMD:0              . if selected then display directory of /CMD files
@D                      . label D
DEVICE                  . if selected then display list of "devices"
```

Here SYSTEM/JCL depends on label in DO command. Call this sample TEST/JCL and

```
  DO TEST (@B) produces     DIR /BAS:0 (A=N,P)
  DO TEST (@C) produces     DIR /CMD:0
```

and so on; if no label is specified, then only initial lines before first label go into SYSTEM/JCL

```
  DO TEST produces          . Labeled choices
                            CLS
                            TIME (CLOCK)
```

INDEX

J C L   I N S T R U C T I O N S

In this column the first page number refers to the detailed reference section. Other page numbers (if any) may refer to discussion in the "Outline" section. The capital letter "C" indicates "compilation" instructions (instructions without "C" are "executable").

BIBLIOGRAPHY

LDOS Manual for Model I and III
by Logical Systems, Inc.

Disk System Owner's Manual Model 4/4P
by Radio Shack

Mod III by Chris for LDOS 5.3
by Christopher Fara
published by Computer News 80

G E N E R A L   I N D E X