Job Control Language

Job Control Language (**JCL**) is a name for scripting languages used on IBM mainframe operating systems to instruct the system on how to run a batch job or start a subsystem.[1]

More specifically, the purpose of JCL is to say which programs to run, using which files or devices [2] for input or output, and at times to also indicate under what conditions to skip a step.

There are two distinct IBM Job Control languages:

- one for the operating system lineage that begins with DOS/360 and whose latest member is z/VSE; and
- the other for the lineage from OS/360 to z/OS, the latter now including JES extensions, Job *Entry* Control Language (JECL).

They share some basic syntax rules and a few basic concepts, but are otherwise very different.[3] The VM operating system does not have JCL as such; the CP and CMS components each have command languages.

Contents

Terminology

Certain words or phrases used in conjunction to JCL are specific to IBM mainframe technology.

- Dataset: a "dataset" is a file; it can be temporary or permanent, and located on a disk drive, tape storage, or other device.[4][5]
- Member: a "member" of a partitioned dataset (PDS) is an individual dataset within a PDS. A member can be accessed by specifying the name of the PDS with the member name in parentheses. For example, the system macro GETMAIN in SYS1.MACLIB can be referenced as SYS1.MACLIB(GETMAIN).[6]
- Partitioned dataset: a "partitioned dataset" or PDS is collection of members, or archive, typically used to represent system libraries. As with most such structures, a member, once stored, cannot be updated; the member must be deleted and replaced, such as with the IEBUPDTE utility.[6] Partitioned datasets are roughly analogous to ar-based static libraries in Unix-based systems.
- USS: Unix system services, a Unix subsystem running as part of MVS, and allowing Unix files, scripts, tasks, and programs to run on a mainframe in a UNIX environment.

Motivation

Originally, mainframe systems were oriented toward batch processing. Many batch jobs require setup, with specific requirements for main storage, and dedicated devices such as magnetic tapes, private disk volumes, and printers set up with special forms.[7] JCL was developed as a means of ensuring that all required resources are available before a job is scheduled to run. For example, many systems, such as Linux allow identification of required datasets to be specified on the command line, and therefore subject to substitution by the shell, or generated by the program at run-time. On these systems the operating system job scheduler has little or no idea of the requirements of the job. In contrast, JCL explicitly specifies all required datasets and devices. The scheduler can pre-allocate the resources prior to releasing the job to run. This helps to avoid "deadlock", where job A holds resource R1 and requests resource R2, while concurrently running

job B holds resource R2 and requests R1. In such cases the only solution is for the computer operator to terminate one of the jobs, which then needs to be restarted. With job control, if job A is scheduled to run, job B will not be started until job A completes or releases the required resources.

Features common to DOS and OS JCL

Jobs, steps and procedures

For both DOS and OS the unit of work is the job. A job consists of one or several steps, each of which is a request to run one specific program. For example, before the days of relational databases, a job to produce a printed report for management might consist of the following steps: a user-written program to select the appropriate records and copy them to a temporary file; sort the temporary file into the required order, usually using a general-purpose utility; a user-written program to present the information in a way that is easy for the end-users to read and includes other useful information such as sub-totals; and a user-written program to format selected pages of the end-user information for display on a monitor or terminal.

In both DOS and OS JCL the first "card" must be the JOB card, which:[8]

- Identifies the job.
- Usually provides information to enable the computer services department to bill the appropriate user department.
- Defines how the job as a whole is to be run, e.g. its priority relative to other jobs in the queue.

Procedures (commonly called *procs*) are pre-written JCL for steps or groups of steps, inserted into a job. Both JCLs allow such procedures. Procs are used for repeating steps which are used several times in one job, or in several different jobs. They save programmer time and reduce the risk of errors. To run a procedure one simply includes in the JCL file a single "card" which copies the procedure from a specified file, and inserts it into the jobstream. Also, procs can include *parameters* to customize the procedure for each use.

Basic syntax

Both DOS and OS JCL have a maximum usable line length of 80 characters, because when DOS/360 and OS/360 were first used the main method of providing new input to a computer system was 80-column punched cards.[9] It later became possible to submit jobs via disk or tape files with longer record lengths, but the operating system's job submission components ignored everything after character 80.

Strictly speaking both operating system families use only 71 characters per line. Characters 73-80 are usually card sequence numbers which the system printed on the end-of-job report and are useful for identifying the locations of any errors reported by the operating system. Character 72 is usually left blank, but it can contain a nonblank character to indicate that the JCL statement is continued onto the next card.

All commands, parameter names and values have to be in capitals, except for USS filenames.

All lines except for in-stream input (see below) have to begin with a slash "`/`", and all lines which the operating system processes have to begin with two slashes `//` - always starting in the *first column*. However, there are two exceptions: the delimiter statement and the comment statement. A delimiter

statements begins with a slash and an asterisk (`/*`), and a comment statement in OS JCL begins with a pair of slashes and asterisk (`//*`) or an asterisk in DOS JCL.

Many JCL statements are too long to fit within 71 characters, but can be extended on to an indefinite number of continuation cards by:

OS JCL	DOS JCL
Ending all actual JCL cards except the last at a point where the syntax requires a comma (`,`)	Ending all actual JCL cards except the last at a point where the syntax requires a comma (`,`) and a non-blank character in column 72
Starting each continuation card with `//` in column 1 and then at least 1 space	Starting each continuation card with spaces and continuing in column 15

The structure of the most common types of card is:[10]

OS JCL	DOS JCL
▪ `//` ▪ Name field for this statement, following `//` with no space between. If this statement does not have a name at least one blank immediately follows the `//`. ▪ Space(s) ▪ Statement type ▪ Space(s) ▪ Parameters, which vary depending on the statement type, separated by commas and with no space between them.	▪ `//` (spaces if this is a continuation of a previous line) ▪ Statement type for this statement, following `//` with a space between. ▪ Space(s) ▪ Name of resource ▪ Space(s) ▪ Parameters, which vary depending on the statement type, separated by commas and with no space between them. Positional parameters, followed by keyword parameters.

In-stream input

DOS and OS JCL both allow in-stream input, i.e. "cards" which are to be processed by the application program rather than the operating system. Data which is to be kept for a long time will normally be stored on disk, but before the use of interactive terminals became common the only way to create and edit such disk files was by supplying the new data on cards.

DOS and OS JCL have different ways of signaling the start of in-stream input, but both end in-stream input with `/*` at column 1 of the card following the last in-stream data card. This makes the operating system resume processing JCL in the card following the `/*` card.[11]

- OS JCL: DD statements can be used to describe in-stream data, as well as data sets. A DD statement dealing with in-stream data has an asterisk (*) following the DD identifier, e.g. `//SYSIN DD *`. JCL statements can be included as part of in-stream data by using the DD DATA statements.

  An operand named **DLM** allowed specifying a delimiter (default is "/*"). Specifying an alternate delimiter allows JCL to be read as data, for example to copy procedures to a library member or to submit a job to the *internal reader*.

  - An example,[12] which submits a job to the *Internal Reader* (**INTRDR**) and then deletes two files is:

```
//SUBM     EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=Z
//SYSUT2   DD   SYSOUT=(A,INTRDR)
//SYSIN    DD   DUMMY
//SYSUT1   DD   DATA,DLM=ZZ
//RUNLATR  JOB  ACCT,MANIX,CLASS=A.TYPRUN=HOLD
//* ^ a JOB to run later
//CPUHOG   EXEC PGM=PICALC1K
//OUTPUT   DD   DSN=PICALC.1000DGTS,SPACE=(TRK,1),DISP=(,KEEP)
ZZ
//* ^ as specified by DLM=ZZ
//DROPOLDR EXEC PGM=IEFBR14
//DELETE4  DD   DSN=PICALC.4DGTS,DISP=(OLD,DELETE)
//DELETE5  DD   DSN=PICALC.5DGTS,DISP=(OLD,DELETE)
```

1. The program called PICALC1K will await (TYPRUN=HOLD) being released manually
2. The program called IEFBR14 will run NOW and upon completion, the two existing files, PICALC.4DGTS and PICALC.5DGTS will be deleted.

- DOS JCL: Simply enter the in-stream data after the EXEC card for the program.

Complexity

Much of the complexity of OS JCL, in particular, derives from the large number of options for specifying dataset information. While files on Unix-like operating systems are abstracted into arbitrary collections of bytes, with the details handled in large part by the operating system, datasets on OS/360 and its successors expose their file types and sizes, record types and lengths, block sizes, device-specific information like magnetic tape density, and label information. Although there are system defaults for many options, there is still a lot to be specified by the programmer, through a combination of JCL and information coded in the program. The more information coded in the program, the less flexible it is, since information in the program overrides anything in the JCL; thus, most information is usually supplied through JCL.

For example, to copy a file on Unix operating system, the user would enter a command like:

```
cp oldFile newFile
```

The following example, using JCL, might be used to copy a file on OS/360:

```
//IS198CPY JOB (IS198T30500),'COPY JOB',CLASS=L,MSGCLASS=X
//COPY01   EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=OLDFILE,DISP=SHR
//SYSUT2   DD DSN=NEWFILE,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(40,5),RLSE),
//            DCB=(LRECL=115,BLKSIZE=1150)
//SYSIN  DD DUMMY
```

A second explanation for the complexity of JCL is the different expectations for running a job from those found in a PC or Unix-like environment.

- Low-end System/360 CPUs were less powerful and more expensive than the mid-1980s PCs for which MS-DOS was designed. OS/360 was intended for systems with a minimum memory size of 32 KB and DOS/360 for systems with a minimum of 16 KB. A 360/30 CPU—low-end when System/360 was announced in 1964—processed 1.8K to 34.5K instructions

per second.[13] The first IBM PC in 1981 had 16 KB or 64 KB of memory and would process about 330K instructions per second.[14][15] As a result, JCL had to be easy for the *computer* to process, and ease of use by programmers was a much lower priority. In this era, programmers were much cheaper than computers.

- JCL was designed for batch processing. As such, it has to tell the operating system everything, including what to do depending on the result of a step. For example, `DISP=(NEW,CATLG,DELETE)` means "if the program runs successfully, create a new file and catalog it; otherwise delete the new file." Programs run on a PC frequently depend on the user to clean up after processing problems.
- System/360 machines were designed to be shared by all the users in an organization. So the `JOB` card tells the operating system how to bill the user's account (`IS198T30500`), what predefined amount of storage and other resources may be allocated (`CLASS=L`), and several other things. `//SYSPRINT DD SYSOUT=*` tells the computer to print the program's report on the default printer which is loaded with ordinary paper, not on some other printer which might be loaded with blank checks. `DISP=SHR` tells the operating system that other programs can read `OLDFILE` at the same time.

Later versions of the DOS/360 and OS/360 operating systems retain most features of the original JCL—although some simplification has been made, to avoid forcing customers to rewrite all their JCL files. Many users save as a *procedure* any set of JCL statements which is likely to be used more than once or twice.[16]

The syntax of OS JCL is similar to the syntax of macros in System/360 assembly language, and would therefore have been familiar to programmers at a time when many programs were coded in assembly language.

DOS JCL

Positional parameters

```
//TLBL TAPEFIL,'COPYTAPE.JOB',,,,2
//ASSGN SYS005,200
//DLBL DISKFIL,'COPYTAPE.JOB',0,SD
//EXTENT SYS005,VOL01,1,0,800,1600
```

DOS JCL parameters are positional, which makes them harder to read and write, but easier for the system to parse.

- The programmer must remember which item goes in which position in every type of statement.
- If some optional parameters are omitted but later ones are included, the omitted parameters must be represented by commas with no spaces, as in the TLBL statement above.

DOS JCL to some extent mitigates the difficulties of positional parameters by using more statements with fewer parameters than OS JCL. In the example the ASSGN, DLBL and EXTENT statements do the same work (specifying where a new disk file should be stored) as a single `DD` statement in OS JCL.

Device dependence

In the original DOS/360 and in most versions of DOS/VS one had to specify the model number of the device which was to be used for each disk or tape file—even for existing files and for temporary files which would be deleted at the end of the job. This meant that, if a customer upgraded to more modern equipment, many JCL files had to be changed.

Later members of the DOS/360 family reduced the number of situations in which device model numbers were required.

Manual file allocation

DOS/360 originally required the programmer to specify the location and size of all files on DASD. The `EXTENT` card specifies the volume on which the extent resides, the starting absolute track, and the number of tracks. For z/VSE a file can have up to 256 extents on different volumes.

OS JCL

OS JCL consists of three basic statement types:[17]

- `JOB` statement, which identifies the start of the job, and information about the whole job, such as billing, run priority, and time and space limits.
- `EXEC` statement, which identifies the program or *procedure*[18] to be executed in this step of the job,
  and information about the step, including *COND*itions for running or skipping a step.
- `DD` (Data Definition) statements, which identify a data file to be used in a step, and detailed info about that file. `DD` statements can be in any order within the step.

Right from the start, JCL for the OS family (up to and including z/OS) was more flexible and easier to use.

The following examples use the old style of syntax which was provided right from the launch of System/360 in 1964. The old syntax is still quite common in jobs that have been running for decades with only minor changes.

Rules for Coding JCL Statements

Each JCL Statement is divided into five fields:[19]

```
Identifier-Field Name-Field Operation-Field Parameter-Field Comments-Field
               ^          ^               ^               ^
            no space     space          space           space
```

Identifier-Field should be concatenated with *Name-Field,* i.e. there should be no spaces between them.

- *Identifier-Field* (`//`): The identifier field indicates to the system that a statement is a JCL statement rather than data. The identifier field consists of the following:
  - Columns 1 and 2 of all JCL statements, except the delimiter statement, contain `//`
  - Columns 1 and 2 of the delimiter statement contain `/*`
  - Columns 1, 2, and 3 of a JCL comment statement contain `//*`

- *Name-Field*: The name field identifies a particular statement so that other statements and the system can refer to it. For JCL statements, it should be coded as follows:
  - The name must begin in column 3.
  - The name is 1 through 8 alphanumeric or national ($, #, @) characters.
  - The first character must be an alphabetic.
  - The name must be followed by at least one blank.
- *Operation-Field*: The operation field specifies the type of statement, or, for the command statement, the command. *Operation-Field* should be coded as follows:
  - The operation field consists of the characters in the syntax box for the statement.
  - The operation follows the name field.
  - The operation must be preceded and followed by at least one blank.
  - The operation will be one of `JOB`, `EXEC` and `DD`.
- *Parameter-Field*: The parameter field, also sometimes referred to as the operand field, contains parameters separated by commas. Parameter field should be coded as follows:
  - The parameter field follows the operation field.
  - The parameter field must be preceded by at least one blank.
  - The parameter field contains parameters which are keywords that used in the statement to provide information such as the program or dataset name.
- *Comments-Field*: This contains comments. *Comments-Field* should be coded as Follows:
  - The comments field follows the parameter field.
  - The comments field must be preceded by at least one blank.

Keyword parameters

```
//NEWFILE DD DSN=MYFILE01,UNIT=DISK,SPACE=(TRK,80,10),
//           DCB=(LRECL=100,BLKSIZE=1000),
//           DISP=(NEW,CATLG,DELETE)
```

All of the major parameters of OS JCL statements are identified by keywords and can be presented in any order. A few of these contain two or more sub-parameters, such as `SPACE` (how much disk space to allocate to a new file) and `DCB` (detailed specification of a file's layout) in the example above. Sub-parameters are sometimes positional, as in `SPACE`, but the most complex parameters, such as `DCB`, have keyword sub-parameters.

Positional parameter must precede keyword parameters. Keyword parameters always assign values to a keyword using the equals sign (=).[20]

Data access (DD statement)

The `DD` statement is used to reference data. This statement links a program's internal description of a dataset to the data on external devices: disks, tapes, cards, printers, etc. The DD may provide information such as a device type (e.g. '181','2400-5','TAPE'), a volume serial number for tapes or disks, and the description of the data file, called the `DCB` subparameter after the Data Control Block (DCB) in the program used to identify the file.

Information describing the file can come from three sources: The DD card information, the dataset label information for an existing file stored on tape or disk, and the DCB macro coded in the program. When the file is opened this data is merged, with the DD information taking precedence over the label information, and the DCB information taking precedence over both. The updated description is then written back to the dataset label. This can lead to unintended consequences if incorrect DCB information is provided.[21]

Because of the parameters listed above and specific information for various access methods and devices the DD statement is the most complex JCL statement. In one IBM reference manual description of the DD statement occupies over 130 pages—more than twice as much as the JOB and EXEC statements combined.[22]

The DD statement allows inline data to be injected into the job stream. This is useful for providing control information to utilities such as IDCAMS, SORT, etc. as well as providing input data to programs.

Device independence

From the very beginning, the JCL for the OS family of operating systems offered a high degree of device independence. Even for new files which were to be kept after the end of the job one could specify the device type in generic terms, e.g., `UNIT=DISK`, `UNIT=TAPE`, or `UNIT=SYSSQ` (tape or disk). Of course, if it mattered one could specify a model number or even a specific device address.[23]

Procedures

Procedures permit grouping one or more "*EXEC PGM=*" and *DD* statements and then invoking them with "*EXEC PROC=*procname" -or- simply "EXEC procname" [24]

A facility called a Procedure Library allowed pre-storing procedures.

PROC & PEND

Procedures can also be included in the job stream by terminating the procedure with a `// PEND` statement, then invoking it by name the same was as if it were in a procedure library.

For example:

```
//SUMPRINT PROC
//PRINT    EXEC PGM=IEBGENER
//SYSUT1   DD   DSN=CEO.FILES.DAYEND.RPT24A,DISP=SHR
//SYSUT2   DD   SYSOUT=A
//SYSIN    DD   DUMMY
//         PEND
// EXEC SUMPRINT
```

Parameterized procedures

OS JCL procedures were parameterized from the start, making them rather like macros or even simple subroutines and thus increasing their reusability in a wide range of situations.[25]

```
//MYPROC PROC FNAME=MYFILE01,SPTYPE=TRK,SPINIT=50,SPEXT=10,LR=100,BLK=1000
.....
//NEWFILE DD DSN=&FNAME,UNIT=DISK,SPACE=(&SPTYPE,&SPINIT,&SPEXT),
```

```
//           DCB=(LRECL=&LR,BLKSIZE=&BLK),DISP=(NEW,CATLG,DELETE)
....
```

In this example, all the values beginning with ampersands "&" are parameters which will be specified when a job requests that the procedure be used. The PROC statement, in addition to giving the procedure a name, allows the programmer to specify default values for each parameter. So one could use the one procedure in this example to create new files of many different sizes and layouts. For example:

```
//JOB01  JOB .........
//STEP01 EXEC MYPROC FNAME=JOESFILE,SPTYPE=CYL,SPINIT=10,SPEXT=2,LR=100,BLK=2000
or
//JOB02  JOB .........
//STEP01 EXEC MYPROC FNAME=SUESFILE,SPTYPE=TRK,SPINIT=500,SPEXT=100,LR=100,BLK=5000
```

Referbacks

In multi-step jobs, a later step can use a *referback* instead of specifying in full a file which has already been specified in an earlier step. For example:

```
//MYPROC ..............
//MYPR01 EXEC PGM=.........
//NEWFILE DD DSN=&MYFILE,UNIT=DISK,SPACE=(TRK,50,10),
//           DCB=(LRECL=100,BLKSIZE=1000),DISP=(NEW,CATLG,DELETE)
....
//MYPR02 EXEC PGM=.........
//INPUT01 DD DSN=*.MYPR01.NEWFILE
```

Here, `MYPR02` uses the file identified as `NEWFILE` in step `MYPR01` (`DSN` means "dataset name" and specifies the name of the file; a DSN could not exceed 44 characters[26]).

In jobs which contain a mixture of job-specific JCL and procedure calls, a job-specific step can refer back to a file which was fully specified in a procedure, for example:

```
//MYJOB JOB .........
//STEP01 EXEC MYPROC            Using a procedure
//STEP02 EXEC PGM=.........     Step which is specific to this job
//INPUT01 DD DSN=*.STEP01.MYPR01.NEWFILE
```

where `DSN=*.STEP01.MYPR01.NEWFILE` means "use the file identified as `NEWFILE` in step `MYPR01` of the procedure used by step `STEP01` of this job". Using the name of the step which called the procedure rather than the name of the procedure allows a programmer to use the same procedure several times in the same job without confusion about which instance of the procedure is used in the referback.

Comments

JCL files can be long and complex, and the language is not easy to read. OS JCL allows programmers to include two types of explanatory comment:

- On the same line as a JCL statement. They can be extended by placing a continuation character (conventionally "X") in column 72, followed by "`//  `" in columns 1–3 of the next line.

- Lines which contain only comment, often used to explain major points about the overall structure of the JCL rather than local details. Comment-only lines are also used to divide long, complex JCL files into sections.

```
//MYJOB JOB .........
//*  Lines containing only comments.
//******** Often used to divide JCL listing into sections ********
//STEP01 EXEC MYPROC             Comment 2 on same line as statement
//STEP02 EXEC PGM=.........      Comment 3 has been extended and       X
//         overflows into another line.
//INPUT01 DD DSN=STEP01.MYPR01.NEWFILE
```

Concatenating input files

OS JCL allows programmers to concatenate ("chain") input files so that they appear to the program as *one* file, for example

```
//INPUT01 DD DSN=MYFILE01,DISP=SHR
//         DD DSN=JOESFILE,DISP=SHR
//         DD DSN=SUESFILE,DISP=SHR
```

The 2nd and third statements have no value in the name field, so OS treats them as concatenations. The files must be of the same basic type (almost always sequential), and must have the same record length, however the block length need not be the same.

In early versions of the OS (certainly before OS/360 R21.8) the block length must be in decreasing order, or the user must inspect each instance and append to the named DD statement the maximum block length found, as in, for example,

```
//INPUT01 DD DSN=MYFILE01,DISP=SHR,BLKSIZE=800
//         DD DSN=JOESFILE,DISP=SHR (BLKSIZE assumed to be equal to or less than 800)
//         DD DSN=SUESFILE,DISP=SHR (BLKSIZE assumed to be equal to or less than 800)
```

In later versions of the OS (certainly after OS/MVS R3.7 with the appropriate "selectable units") the OS itself, during allocation, would inspect each instance in a concatenation and would substitute the maximum block length which was found.

A usual fallback was to simply determine the maximum possible block length on the device, and specify that on the named DD statement, as in, for example,

```
//INPUT01 DD DSN=MYFILE01,DISP=SHR,BLKSIZE=8000
//         DD DSN=JOESFILE,DISP=SHR (BLKSIZE assumed to be equal to or less than 8000)
//         DD DSN=SUESFILE,DISP=SHR (BLKSIZE assumed to be equal to or less than 8000)
```

The purpose of this fallback was to ensure that the access method would allocate an input buffer set which was large enough to accommodate any and all of the specified datasets.

Conditional processing

OS expects programs to set a return code which specifies how successful the *program* thought it was. The most common conventional values are:[27]:p.87

- 0 = Normal - all OK
- 4 = Warning - minor errors or problems
- 8 = Error - significant errors or problems
- 12 = Severe error - major errors or problems, the results (e.g. files or reports produced) should not be trusted.
- 16 = Terminal error - very serious problems, do not use the results!

OS JCL refers to the return code as `COND` ("condition code"), and can use it to decide whether to run subsequent steps. However, unlike most modern programming languages, conditional steps in OS JCL are *not* executed if the specified condition is true—thus giving rise to the mnemonic, "If it's true, pass on through [without running the code]." To complicate matters further, the condition can only be specified *after* the step to which it refers. For example:

```
//MYJOB JOB ...........
//STEP01 EXEC PGM=PROG01
....
//STEP02 EXEC PGM=PROG02,COND=(4,GT,STEP01)
....
//STEP03 EXEC PGM=PROG03,COND=(8,LE)
....
//STEP04 EXEC PGM=PROG04,COND=(ONLY,STEP01)
....
//STEP05 EXEC PGM=PROG05,COND=(EVEN,STEP03)
....
```

means:

1. Run `STEP01`, and collect its return code.
2. Don't run `STEP02` if the number 4 is greater than `STEP01`'s return code.
3. Don't run `STEP03` if the number 8 is less than or equal to any previous return code.
4. Run `STEP04` only if `STEP01` abnormally ended.
5. Run `STEP05`, even if `STEP03` abnormally ended.

This translates to the following pseudocode:

```
run STEP01
if STEP01's return code is greater than or equal to  4 then
    run STEP02
end if
if any previous return code is less than  8 then
    run STEP03
end if
if STEP01 abnormally ended then
    run STEP04
end if
if STEP03 abnormally ended then
    run STEP05
else
    run STEP05
end if
```

Note that by reading the steps containing `COND` statements backwards, one can understand them fairly easily. This is an example of logical transposition. However, IBM later introduced IF condition in JCL thereby making coding somewhat easier for programmers while retaining the `COND` parameter (to avoid making changes to the existing JCLs where `COND parm` is used).

The `COND` parameter may also be specified on the `JOB` statement. If so the system "performs the same return code tests for every step in a job. If a JOB statement return code test is satisfied, the job terminates."[28]

Utilities

Jobs use a number of IBM utility programs to assist in the processing of data. Utilities are most useful in batch processing. The utilities can be grouped into three sets:

- Data Set Utilities - Create, print, copy, move and delete data sets.
- System Utilities - Maintain and manage catalogs and other system information.
- Access Method Services - Process Virtual Storage Access Method (VSAM) and non-VSAM data sets.

Difficulty of use

OS JCL is undeniably complex[29] and has been described as "user hostile".[30][31] As one instructional book on JCL asked, "Why do even sophisticated programmers hesitate when it comes to Job Control Language?"[32] The book stated that many programmers either copied control cards without really understanding what they did, or "believed the prevalent rumors that JCL was horrible, and only 'die-hard' computer-types ever understood it" and handed the task of figuring out the JCL statements to someone else.[32] Such an attitude could be found in programming language textbooks, which preferred to focus on the language itself and not how programs in it were run. As one Fortran IV textbook said when listing possible error messages from the WATFOR compiler: "Have you been so foolish as to try to write your own 'DD' system control cards? Cease and desist forthwith; run, do not walk, for help."[33]

Nevertheless, some books that went into JCL in detail emphasized that once it was learned to an at least somewhat proficient degree, one gained freedom from installation-wide defaults and much better control over how an IBM system processed your workload.[32][29] Another book commented on the complexity but said, "take heart. The JCL capability you will gain from [the preceding chapter] is all that most programmers will ever need."[29]

Job Entry Control Language

On IBM mainframe systems *Job Entry Control Language* or *JECL* is the set of command language control statements that provide information for the spooling subsystem – JES2 or JES3 on z/OS or VSE/POWER for z/VSE. JECL statements may "specify on which network computer to run the job, when to run the job, and where to send the resulting output."[27]

JECL is distinct from job control language (JCL), which instructs the operating system *how* to run the job.

There are different versions of JECL for the three environments.

OS/360

An early version of Job Entry Control Language for OS/360 Remote Job Entry (Program Number 360S-RC-536) used the identifier `. .` in columns 1–2 of the input record and consisted of a single control statement: `JED` (Job Entry Definition). "Workstation Commands" such as `LOGON`, `LOGOFF`, and

STATUS also began with . . .[34]

pre-JES JECL

Although the term had not yet been developed, HASP did have similar functionality to what would become the JECL of JES, including /* syntax.

z/OS

For JES2 JECL statements start with /*, for JES3 they start with //*, except for remote /*SIGNON and /*SIGNOFF commands. The commands for the two systems are completely different.

JES2 JECL

The following JES2 JECL statements are used in z/OS 1.2.0.[35]

JECL statement	Function	Example
/*$*command*	Enters an operator (console) command	/*$S PRINTER3[36]
/*JOBPARM	Specifies values for job-related parameters	/*JOBPARM TIME=10
/*MESSAGE	Sends a message to the operator console	/*MESSAGE CALL JOE AT HOME IF JOB ABENDS
/*NETACCT	Specifies account number for network job	/*NETACCT 12345
/*NOTIFY	Specifies destination for notification messages	/*NOTIFY SAM
/*OUTPUT	Specifies *SYSOUT* dataset options	/*OUTPUT FORMS=BILL
/*PRIORITY	Sets job selection priority	/*PRIORITY 15
/*ROUTE	Specifies output destination or execution node	/*ROUTE PRT RMT5
/*SETUP	Requests volume mounting or other offline operation	/*SETUP TAPE01,TAPE02
/*SIGNOFF	Ends remote session	/*SIGNOFF
/*SIGNON	Begins remote session	/*SIGNON REMOTE5 *password*
/*XEQ	Specifies execution node	/*XEQ DENVER
/*XMIT	Indicates job or dataset to be transmitted to another network node	/*XMIT NYC

JES3 JECL

The following JES3 JECL statements are used in z/OS 1.2.0[37]

JECL statement	Function	Example
`//**`*command*	Enters a JES3 operator (console) command	
`//*DATASET`	Marks the beginning of an in-stream dataset	
`//*ENDDATASET`	Marks the end of an in-stream dataset	
`//*ENDPROCESS`	Marks the end of a series of `//*PROCESS` statements	
`//*FORMAT`	Specifies `SYSOUT` dataset options	
`//*MAIN`	Specifies values for job-related parameters	
`//*NET`	Identifies relationships among jobs using JES3 *dependent job control*	
`//*NETACCT`	Specifies account number for network job	
`//*OPERATOR`	Sends a message to the operator console	
`//*PAUSE`	Stops the input reader	
`//*PROCESS`	Identifies a non-standard job	
`//*ROUTE`	Specifies the execution node for the job	
`/*SIGNOFF`	Ends remote session	`/*SIGNOFF`
`/*SIGNON`	Begins remote session	

z/VSE

For VSE JECL statements start with '`*  $$`' (note the *single* space). The Job Entry Control Language defines the start and end lines of JCL jobs. It advises VSE/POWER how this job is handled. JECL statements define the job name (used by VSE/POWER), the class in which the job is processed, and the disposition of the job (i.e. `D`, `L`, `K`, `H`).

JECL statement[38]	Function	Example
`* $$ CTL`	Establishes a default *input class*	`* $$ CTL CLASS=A`
`* $$ JOB`	Specifies attributes of a job	`* $$ JOB JNM=PYRL,PRI=9`
`* $$ EOJ`	Marks the end of a job	`* $$ EOJ`
`* $$ RDR`	Inserts a file from a 3540 diskette into the input stream	`* $$ RDR SYS005,'fname',2`
`* $$ PRT`	Specifies characteristics of spooled print files "LST' is a synonym for "PRT"	`* $$ PRT FNO=STD,COPY=2`
`* $$ PUN`	Specifies characteristics of spooled punch files	`* $$ PUN DISP=T,TADDR=280`
`* $$ SLI`	Inserts data ("book") from source statement library into the input stream	`* $$ SLI A.JCL1`
`* $$ DATA`	Inserts data from the card reader into a book retrieved from the source statement library	`* $$ DATA INPUT1`

Example:

```
* $$ JOB JNM=NAME,DISP=K,CLASS=2
```

```
[some JCL statements here]

* $$ EOJ
```

Other systems

Other mainframe batch systems had some form of job control language, whether called that or not; their syntax was completely different from IBM versions, but they usually provided similar capabilities. Interactive systems include "command languages"—command files (such as PCDOS ".bat" files) can be run non-interactively, but these usually do not provide as robust an environment for running unattended jobs as JCL. On some computer systems the job control language and the interactive command language may be different. For example, TSO on z/OS systems uses CLIST or Rexx as command languages along with JCL for batch work. On other systems these may be the same.

See also

- dd (Unix), Unix program inspired by `DD`
- IBM mainframe utility programs
- Batch processing
- Data set (IBM mainframe)#Generation Data Group

References

1. "Every job submitted for execution ... must include JCL statements" -- ibm.com
2. and many more complex details, such as whether the file is to be retained or deleted, the maximum of disk space to which it can grow, the name of a tape to be pre-mounted
3. Ashley and Fernandez, *Job Control Language*, p. 1.
4. Ashley and Fernandez, *Job Control Language*, p. 5.
5. McQuillen, *System/360–370 Assembler Language*, pp. 385–386.
6. McQuillen, *System/360–370 Assembler Language*, pp. 288–289, 400.
7. McQuillen, *System/360–370 Assembler Language*, pp. 22–24.
8. McQuillen, *System/360–370 Assembler Language*, pp. 380–382.
9. Stern and Stern, *Structured COBOL Programming*, pp. 528–529.
10. Stern and Stern, *Structured COBOL Programming*, pp. 529, 531.
11. Stern and Stern, *Structured COBOL Programming*, pp. 529, 537.
12. modeled on https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.hasc300/has using knowledge dating back to when Green Cards came from IBM, and Manix worked for a company owning an IBM card sorter
13. "IBM Archives: System/360 Model 30" (http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP2030.html). *www-03.ibm.com*. 2003-01-23. Retrieved 2016-04-25.
14. IBM PC (http://www.vintage-computer.com/ibm_pc.shtml)
15. IBM-compatible computers (http://www3.iath.virginia.edu/elab/hfl0108.html) History of PCs Archived (https://web.archive.org/web/20070814010809/http://www3.iath.virginia.edu/elab/hfl0108.html) August 14, 2007, at the Wayback Machine
16. Brown, Gary DeWard (2002). *zOS JCL* (https://books.google.com/books?id=K8kMJa8arIIC&pg=PA248) (fifth ed.). John Wiley & Sons. p. 248. ISBN 0471-236357.

17. Ashley and Fernandez, *Job Control Language*, pp. 8, 23. There are also two additional statements, PROC and PEND, used to test JCL procedures.
18. A pre-stored set of "EXEC PGM=" and "DD" JCL commands which could be parameterized
19. Ashley and Fernandez, *Job Control Language*, pp. 12–16.
20. Ashley and Fernandez, *Job Control Language*, pp. 13–15.
21. IBM Corporation (August 1978). *OS/VS MVS Data Management Services Guide* (http://www.prycroft6.com.au/misc/download/GC26-3875-0_MVS_DataMgmtSrvcsGde_Aug78OCR.pdf) (PDF). Retrieved Oct 17, 2014.
22. IBM Corporation (June 1971). *IBM System/360 Operating System: Job Control Language Reference* (http://www.bitsavers.org/pdf/ibm/360/os/R20.1_Mar71/GC28-6704-1_OS_JCL_Reference_Rel_20.1_Jun71.pdf) (PDF). Retrieved June 25, 2019.
23. McQuillen, *System/360–370 Assembler Language*, pp. 297, 406–407.
24. the default for the EXEC statement is PROC=
25. Ashley and Fernandez, *Job Control Language*, pp. 129–131.
26. "Data set names" (https://www.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.idad400/name.htm). "Data set names must not exceed 44 characters, including all name segments and periods."
27. Brown, Gary DeWard (2002). *zOS JCL* (https://books.google.com/books?id=K8kMJa8arlIC&dq=jecl+definition&pg=PA3). John Wiley & Sons. ISBN 9780471426738. Retrieved 2014-05-05.
28. IBM Corporation. "Relationship of the COND parameters on JOB and EXEC statements" (https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.ieab500/iea3b5_Relationship_of_the_COND_parameters_on_JOB_and_EXEC_statements.htm). *IBM Knowledge Center*. Retrieved Feb 21, 2018.
29. McQuillen, *System/360–370 Assembler Language*, pp. 406–407.
30. Charley, Alfred (1993). *NetView: IBM's Network Management Product*. New York: Van Nostrand Reinhold. p. 93 (https://books.google.com/books?id=TPVSAAAAMAAJ&q=jcl+%22user+hostile%22&dq=jcl+%22user+hostile%22&hl=en&sa=X&ved=2ahUKEwj3m6va2-vkAhXtkOAKHVHIDRUQ6AEwAXoECAIQAg). ISBN 0-442-01407-4.
31. Mathew W. Blode (April 6, 2020). "Newly unemployed New Yorkers are being frustrated by 1970s-era technology(nytimes.com)" (https://hn.matthewblode.com/item/22785474). Retrieved May 7, 2020. "JCL in particular is notoriously user hostile and has been called "the worst programming language ever designed" by Fred Brooks ... (http://dtsc.dfw.ibm.com/MVSDS/'HTTPD2.APPS.ZOSCLASS.PDF(ZCLA...)." {{cite web}}: External link in |quote= (help)
32. Ashley and Fernandez, *Job Control Language*, pp. vii–viii, back cover.
33. Blatt, John M. (1971). *Introduction to FORTRAN IV Programming: Using the WATFOR/WATFIV Compilers*. Pacific Palisades, California: Goodyear Publishing Company. p. 276. ISBN 0-87620-440-X.
34. IBM Corporation (1968). *IBM System/360 Operating System Remote Job Entry* (http://bitsavers.informatik.uni-stuttgart.de/pdf/ibm/360/rje/C30-2006-1_Remote_Job_Entry_May68.pdf) (PDF). Retrieved 2014-05-05.
35. IBM Corporation. "Job Entry Subsystem 2 (JES2) Control Statements" (http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/iea2b510/1.1.2?SHELF=EZ2ZO103&DT=20010626133938&CASE=). *z/OS V1R2.0 MVS JCL*. Retrieved February 25, 2013.
36. other examples can be viewed at Houston Automatic Spooling Priority#Operator Commands
37. IBM Corporation. "Job Entry Subsystem 3 (JES3) Control Statements" (http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/iea2b510/1.1.2?SHELF=EZ2ZO103&DT=20010626133938&CASE=). *z/OS V1R2.0 MVS JCL*. Retrieved February 25, 2013.

38. IBM Corporation (1974). *DOS/VS POWER/VS Installation and Operations* (http://bitsavers.trailing-edge.com/pdf/ibm/370/DOS_VS/GC33-5403-1_DOS_VS_POWER_VS_Installation_and_Operation_Sep74.pdf) (PDF).

Sources

- "z/OS V1R6.0 MVS JCL User's Guide" (http://publibz.boulder.ibm.com/epubs/pdf/iea2b540.pdf) (PDF) (5th ed.). IBM. September 2004.
- "z/OS V1R7.0 MVS JCL Reference" (http://publibz.boulder.ibm.com/epubs/pdf/iea2b661.pdf) (PDF) (11th ed.). IBM. April 2006.
- Johnston, Jerry (1 April 2005). "VSE: A Look at the Past 40 Years" (https://web.archive.org/web/20090304014628/http://www.zjournal.com/index.cfm?section=article&aid=293). *z/Journal*. Thomas Communications. Archived from the original (http://www.zjournal.com/index.cfm?section=article&aid=293) on 4 March 2009.
- "Computer Chronicles: 1972 - 1981" (https://web.archive.org/web/20090621080529/http://library.thinkquest.org/22522/timeline3_en.html). *ThinkQuest*. Oracle Corporation. 1998. Archived from the original (http://library.thinkquest.org/22522/timeline3b_en.html) on 21 June 2009.
- DeWard Brown, Gary (7 June 2002). *zOS JCL* (5th ed.). Wiley. ISBN 978-0-471-23635-1.
- "JCL Statement Fields" (http://publib.boulder.ibm.com/infocenter/zos/v1r11/index.jsp?topic=/com.ibm.zos.r11.ieab600/iea2b69009.htm). *z/OS V1R11.0 MVS JCL Reference z/OS V1R10.0-V1R11.0*. IBM. 2010.
- IBM Corporation (March 2007). *Introduction to the New Mainframe: z/VSE Basics* (http://www.redbooks.ibm.com/redbooks/pdfs/sg247436.pdf) (PDF). ISBN 978-0-73-848624-6. Retrieved 2017-12-06.
- Ashley, Ruth; Fernandez, Judi N. (1978). *Job Control Language: A Self-Teaching Guide*. New York: John Wiley & Sons. ISBN 0-471-03205-0.
- McQuillen, Kevin (1975). *System/360–370 Assembler Language (OS)*. Fresno, California: Mike Murach & Associates. LCCN 74-29645 (https://lccn.loc.gov/74-29645).
- Stern, Nancy; Stern, Robert A. (1980). *Structured COBOL Programming* (https://archive.org/details/structuredcobolp0000ster_v3q6) (3rd ed.). New York: John Wiley & Sons. ISBN 0-471-04913-1.