jBASE Command Language

Programmers Reference Manual
jBASE 4.1

Contents

Documentation Conventions

This manual uses the following conventions:

Convention	Usage
BOLD	In syntax, bold indicates commands, function names, and options. In text, bold indicates keys to press, function names, menu selections, and MS-DOS commands.
UPPERCASE	In syntax, uppercase indicates JBASE commands, keywords, and options; BASIC statements and functions; and SQL statements and keywords. In text, uppercase also indicates JBASE identifiers such as filenames, account names, schema names, and Windows NT filenames and pathnames.
UPPERCASE *Italic*	In syntax, italic indicates information that you supply. In text, italic also indicates UNIX commands and options, filenames, and pathnames.
`Courier`	Courier indicates examples of source code and system output.
`Courier Bold`	**Courier Bold** In examples, courier bold indicates characters that the user types or keys (for example, <Return>).
[]	Brackets enclose optional items. Do not type the brackets unless indicated.
{}	Braces enclose nonoptional items from which you must select at least one. Do not type the braces.
ItemA | itemB	A vertical bar separating items indicates that you can choose only one item. Do not type the vertical bar.
. . .	Three periods indicate that more of the same type of item can optionally follow.
⇒	A right arrow between menu options indicates you should choose each option in sequence. For example, “Choose **File ⇒Exit**” means you should choose **File** from the menu bar, and then choose **Exit** from the File pull-down menu.

Syntax definitions and examples are indented for ease in reading.

All punctuation marks included in the syntax—for example, commas, parentheses, or quotation marks—are required unless otherwise indicated.

Syntax lines that do not fit on one line in this manual are continued on subsequent lines. The continuation lines are indented. When entering syntax, type the entire syntax entry, including the continuation lines, on the same input line.

CHAPTER 1 JCOMMAND LANGUAGE

This manual demonstrates to the end user how to write and execute jCL (jBASE Command Language) programs. It also discusses how jCL manipulates data in the various buffers and selects registers.
Most of the following text assumes that you will be using the functionally superior PQN variant of the jCL language. The PQ and PQN Differences section discusses the differences between the two variants.

JCL PROGRAM STRUCTURE

A jCL program is stored as a text record in a file where the first line is usually a PQ or PQN, unless you have elected to run the program as a UNIX executable (script), in which case the first line will contain #!usr/jbc/bin/jpq.
Subsequent lines contain jCL statements, which can execute other programs, manipulate data in the buffers, and control the flow of program execution and so on.
jCL program statements comprise a numeric label (optional), a command and any command arguments.
There are numerous facilities, which enable you to control program flow and to call (and return) or jump to other jCL programs. You can also construct internal or external subroutines.

Labels

Labels appear at the start of a line and are always numeric (1, 330, 1000, etc.). You should always have at least one space between the label and the statement.

Grouping commands

You can place more than one command on a line by separating each command with a subvalue mark character (x"FC" - entered by typing <ctrl \>), which are executed sequentially, left to right.
As some commands require one or more dedicated lines in the program grouping is not possible on the same line. These commands are:

( )	[]	B	BO	F	F
FB	F-CLEAR	F-FREE	F-KLOSE	F-OPEN	F-READ
F-UREAD	F-WRITE	GOSUB	H	IBH	IH
MVA	MVD	O	P	RI	RO

RSUB RTN U X

Lines, which are a continuation of a T or L command from a previous line, are not suitable for grouping.

Other commands can follow the M (Mark) on the same line, but it must be the first command on a line. A subvalue mark must separate commands following the M command, on the same line rather than a space (unlike numeric labels).

Comment Lines

A command line which starts with a "C" or an "*" are treated as comments.
If you include a comment in a multiple command line, it ignores anything between the "C" or "*" following a subvalue mark (or the end of the line).
A label remains active if the "C" or "*" is placed after it on the line.

Executing jCL Programs

You can execute jCL programs in several ways:

- enter the name of the program from jSHELL,
- "jump to" another jCL program of the same type by using the ( ) command,
- "call" another jCL program of the same type, as a subroutine, by using the [] command
- use a PERFORM, EXECUTE or CHAIN statement from a jBASIC program, or
- convert the program to a UNIX executable and call it from any shell. Change the first line to #!usr/jbc/bin/jpq and then use chmod to create an executable file.

Once you start a jCL program, it will remain active until:

- control is explicitly passed to another jCL program
- the jCL program is explicitly exited
- all of the lines of the jCL program are exhausted
- encounters a fatal error.

Even when the jCL program temporarily passes control to another process such as jED or jBASIC, it will remain in control (unless control is passed to a jBASIC program which then CHAINs or ENTERs another jCL program). Exiting from the called process will return control to the jCL program.
If you do not want to store the main body of your jCL program in the MD file, you can create a "pointer" jCL program in the MD. For example, to run a jCL program called DAILY held in a file called REPORTS, create an MD entry as follows:

```
DAILYYREPORT
001 PQN
002 (REPORTS DAILY)
```

This will chain to the jCL program DAILY in file REPORTS.

NOTE: that the "pointer" program and the "pointed to" program can have the same name.

Login jCL Programs

You can create jCL programs, which run automatically, each time a user logs in or logs in to a specific account. Simply create a jCL program with the same name as the account or user, and put "jshell -" on the last line of the login script (usually.profile).
Use this feature for example, to configure the jSHELL environment, implement a security system, or display an initial menu.

EXAMPLE

Let us assume that all users who login to an account called SALES view an initial menu, which is a jBASIC program, called MENU held in a file called PROGS. You would create a jCL program in the MD of the SALES account and call it SALES:

```
SALES
001 PQN
002 HRUN PROGS MENU
003 P
```

Each time a user logs in to the SALES account, the system will search for a program called SALES in the MD of the account. The jCL program will then run the MENU program from the PROGS file.

INTERNAL DATA STORAGE

Command language programs use a series of buffers to store and manipulate data internally and use these buffers in much the same way as variables in a jBASIC program.

The four main classes of buffers are:

1. "input" and "output",

and sub-classed as

2. "primary" and "secondary".

The provided nine file buffers allow the movement of data to and from file records. There is also a "fast" file buffer to give you quick access to records.
In addition to the main input/output and file buffers, there is a range of five "select" buffers used to manipulate, lists (typically record keys).
You will need a thorough understanding of the way in which these buffers work to write (or maintain) anything other than simple jCL programs.

The four jCL main buffers are:

- primary input buffer (PIB)
- secondary input buffer (SIB)
- primary output buffer (POB)
- secondary output buffer (SOB)

1. The input buffers receive and manipulate data;
2. The output buffers issue shell commands to the system.
3. Most of the time you will use primary buffers as the active buffers as very few jCL commands use the secondary buffers (particularly the secondary input buffer).

Primary Input Buffer

The primary input buffer is a work area used to collect terminal input or for temporary storage of control parameters.
Reference individual parameters within the PIB by preceding the parameter number with a percent sign (for example, %3)

When you start a jCL program, the PIB will contain exactly as entered the jCL program name and any supplied arguments.
When you execute a jCL program from a jBASIC CHAIN, EXECUTE/PERFORM statement, the PIB will contain the CHAIN, EXECUTE or PERFORM command.
If you execute a jBASIC program from jCL, you can access the contents of the PIB from jBASIC with the PROCREAD and PROCWRITE statements. Control returns to the jCL program when the jBASIC program ends - unless passing control to a jBASIC program, which then CHAINs or ENTERs another jCL program.

Secondary Input Buffer

The secondary input buffer has three main roles:

- collection of input from the terminal when using an IN command
- collection of error codes generated from executed programs and jQL queries (see the IF E command), and
- collection of spooler hold file numbers in the form Entry # n, where n is the hold file number.

Each time you use one of these facilities it overwrites the data in the SIB.
When the primary input buffer is active, use the MS command to copy the entire contents of the secondary input buffer to the primary input buffer.

Primary Output Buffer

The primary output buffer builds shell commands submitted for processing when executing the P command. With the exception of other jCL programs and the LOGTO command, you can construct and execute any shell command in this way.
You should always use a jCL transfer function when invoking one jCL program from another.
Automatically placed at the end of the command in the POB when executed by the P command is a carriage return.

Secondary Output Buffer

The secondary output buffer provides a "stack" of responses, which are fed into interactive processes, such as jBASIC or jED, when they request input.
The SOB becomes active when you use the STON (stack on) command. You can then place data in the SOB before say, execution of a jBASIC program.

The use of stacked data is to respond to INPUT requests, instead of taking input from the keyboard - rather like the DATA statement in a jBASIC program.

NOTE: the same internal storage area uses both the SOB and the jBASIC DATA statement. If a CHAIN, EXECUTE or PERFORM statement in a jBASIC program calls your jCL program, jCL input statements use any data placed in the stack by a DATA statement.
If executing a jCL program directly from the shell, it clears the SOB before the program begins.

Terminate each line of data placed in the SOB using the H command by typing a less-than character (<) to represent a carriage return.

If you wanted to place more than 140 characters into the buffer, older systems required the use of a line continuation symbol (two less - than characters "<<"). Support for this functionality remains; you do not need to use the line continuation symbol in jBASE.

Active Input Buffer

Many commands access the currently active input buffer ‘A’ rather than referring specifically to the primary or secondary areas. These include the ‘A’ and ‘Ap’ forms of the ‘A’ command, and the ‘B’, ‘D’ (without parameter), ‘F’, ‘IB’ ‘H’, ‘IH’, ‘IP’ and ‘Sp’ commands.
When executing an IN or IBN command, the SIB becomes the active input buffer. Until the execution of a RI, S (n) or MV %n source command the SIB remains active
Use caution when using these commands, also try to structure your programs to minimise the possible confusion between which buffers are currently active. As a minimum, include as many comments as you can.

Active Output Buffer

When executing a jCL program from the shell or issuing a STOFF (Stack Off) command, the POB becomes the active output buffer.
The SOB becomes active only when you use a STON (Stack On) command.
You can refer to the parameters in the active output buffer by using a hash sign (#).
Some operations, such as COPY and EDIT, need two buffers; when creating these commands you will need to switch between the buffers. The Primary buffer holds the command (as it would be entered at the shell prompt) and the secondary buffer holds any further input required by the command. If you fail to specify any required input in the secondary buffer, you will receive a prompt at run-time.

Buffer Data and Pointers

Data held in the input and output buffers are as a series of parameters separated by blanks (PQ) or field marks (PQN).
You can refer directly to a specific parameter by using the parameter's sequence number. For example, you might refer to the 10th parameter as %10.
Alternatively, you can refer to the data in terms of its column (or character) position. For example, you might refer to the 21st column (character) of the primary input buffer as S(21).

Four buffer pointers are used:

- one for each of the active input buffers
- one for each of the output buffers

In the examples, which follow, an up arrow indicates the buffer pointers ↑.

- A carat ^ indicates field marks (the parameter separators in PQN-style jCL programs).
- The input buffer pointer points to a position in the active input buffer. Use the S, F and B commands to move the pointer without affecting the data in the buffer by.
- The S command positions the input buffer pointer to a specific location, while the F and B commands move the pointer forward or backward over the parameters.
- Each output buffer has its own pointer. These pointers always indicate the end of the parameter data. If you move the pointer backwards and write data to the new position, it truncates the original data at the end of the buffer.
- Use the BO command to move the primary output buffer pointer back one parameter, or to clear the secondary output buffer.

File Buffers

Numbered from 1 to 9 jCL provides nine file buffers, used as the medium for transferring data from and to files - reading and writing.
You should always open file buffers before you use them for reading or writing, use also as temporary data storage areas.

The file buffer contains data (like a database record) divided into fields and separated by field marks. If you refer to a field beyond the last field, it creates the required number of additional fields (with null values).
File buffer references are constructed by specifying an ampersand (&), followed by the file buffer number, a period (.) and a numeric value (which might be an indirect reference) to define the field number you want.

For example, if file buffer 4 contains:

```
000 TEST1
001 ABC
002 DEF
003 GHI
004 JKL
```

&4.2 refers to the second field in file buffer 4 - in this case DEF, and &4.0 refers to the record key of the record in file buffer 4 - in this case TEST1.
The record key will be null until you perform a read operation or until you move data into field 0 of the file buffer.
The FB command provides a special "Fast Buffer" facility. You can use FB to read a record from a file into the fast buffer without having to open the file first. There is only one fast buffer. The fields are referred to as &n, where n is the field number. Each time you use the FB command it overwrites existing data.

Select Registers

Use select registers to hold a list of record keys or other multi-valued fields, generated by shell commands that produce lists.
These include

- BSELECT
- ESEARCH
- FORM-LIST
- GET-LIST
- QSELECT
- SEARCH

JBASE Command Language

- SELECT
- SSELECT

Having executed one of the above commands, use the PQ-SELECT and PQ-RESELECT jCL commands to load or reset the select registers.
Numbered from 1 to 5 are five select registers, where each can contain a separate list.
Access the select registers by using an exclamation mark (!) followed by the register number –

EXAMPLE

When accessing a select register, it takes each time the next field from the top of the list.
You cannot manipulate the values in a select register. If you write to a select register, say with the MV command, it replaces the entire list. If you want to use a value more than once, move it to another buffer for temporary storage.

If the Select Register contains

```
KEY1 ^ KEY2 ^ KEY3 ^ KEY4^...
```

The first access will retrieve KEY1, the second KEY2, the third KEY3 and so on.

Buffer References

The use of buffer reference symbols is to make direct or indirect references to the input and output buffers and the file buffers. You can only make direct references to the select registers.

Four special symbols are used:

%	Primary input buffer.
#	Active output buffer.
&	For a file buffer or the "fast" buffer.
!	For a select register.

Referencing the primary input buffer or the active output buffer will not reposition the pointer.
Use the appropriate symbol with a literal value to use direct reference.

% 23	Refers to the 23rd parameter in the primary input buffer

#4	Refers to the 4th parameter in the current output buffer.
&4.19	Refers to field 19 in file buffer 4
!3	Refers to the next value in select register 3.

Use a combination of symbols and literal values to create an indirect reference.

% %3	Refers to the %3th parameter in the primary input buffer If say, %3 contained the value 10, this example would refer to the 10th parameter of the primary input buffer.
&4.%5	Refers to the field in file buffer 4 defined by %5.

You can only make indirect references to the primary input buffer (%), the active output buffer (#) or a file buffer (&). If the resulting value is non-numeric, it uses a zero. In the case of the file buffer (&), you may only use an indirect reference to the parameter, not the file buffer number - for example, &5.%3 is legal, &%3.5 is not.

Use direct or indirect buffer references with the following jCL commands:

( )	F-OPEN,	IBH,	IFN,	MVA
[],	F-READ,	IBP,	IH,	MVD
F;	F-UREAD,	IF,	IP,	RI
F-FREE,	FB,	IF (multivalued),	L,	S
F-CLOSE,	H,	IF (mask),	MV,	T

Branching

Wherever possible, avoid using the obsolete "M"ark command to control branching within the program. Using labels and the GO command will help to preserve clarity and make program maintenance much easier.

Grouping Commands

You can put more than one jCL command on the same line by using a subvalue mark. For example:

```
001 PQN
002 10 T "Enter file name :",+
003 IBP %1
004 F-O 1 %1
005 T "Cannot open ", %1, "..."\ GO 10
```

```
006 T "File ", %1, " opened OK"
```

In this example, if the F-O command cannot open the file, it executes the line immediately following the command (see the F-O command for details); if opened, it executes the next but one line. By grouping an error message and a branch on the "error" line (005), you will avoid the necessity of creating a separate subroutine to handle the error condition.

NOTE that you cannot use grouping like this where a command relies on a particular line structure - the F-O command for example.
You can use a mark with the ctrl \> but it must be the first jCL command on the line. For example, use:

```
M \ IBP:%1 \ IF # %1 X
```

Not:

```
IBP %1 \ M \ IF # %1 X
```

Readability

To increase readability and make it easier to edit and debug your jCL program, use leading blanks to indent lines.
The incorporation of useful comments, using the C or * commands, will help with the future maintenance of your programs.

Time and Date

The following commands provide a simple way of putting the current system time and date into the currently active input buffer:

```
S21
U147F
T
```

Puts the current time, in internal format (seconds since midnight), in the current input buffer (in this case, %21)

```
S10
U147F
D
```

Puts the current date, in internal format (days since 31 Dec 67), in the current input buffer (in this case, %10)

```
S8
U147F
T MTS
```

Puts the current time, in external format (MTS yields hh:mm:ss), in the current input buffer (in this case, %8)

```
S33
U147F
D D2/
```

Puts the current date, in external format (D2/ yields dd/mm/yy or mm/dd/yy - depending on your internationalization settings), in the current input buffer (in this case, %33)

Validating Date and Time

You can use pattern matching to input a valid date or time but it does not catch input like 10/32/94 or 25:25:25.

The example below checks for a valid date in D2/ format by converting it. This mechanism works because an invalid date converts to null.

```
001 PQN
002 10 T "Enter date (mm/dd/yy) :", +
003 IF # %1 Xfinished
004 IBP %1
005 MV %2 "
006 IBH%1;D2/;
007 IF # %2 T B,"Oops!"\ GO 10
008 C Date OK
```

Long Statements

To help with program clarity, you can construct long statements by using several H commands. Make sure there is a space at the end the first H command or before the second (and subsequent) commands.

EXAMPLE

```
001 PQN
002 HGET-LIST listname
003 STON
004 HLIST filename with *A1 EQ”. . .
005 H HEADING "..."
006 P
```

Older systems required the use of a line continuation symbol (two less- than characters "<<").
NOTE: Support for this functionality remains although you do not need to use the line continuation symbol in jBASE.

Concatenation

Use an asterisk (*) to concatenate (join) a series of values. For example:

```
001 PQN
002 MV %2 "string"
003 MV %1 "Text "*%2*" has been concatenated"
004 T %1
```

Will display "Text string has been concatenated"

Spooler Hold File Numbers

If while executing a jCL program it generates a hold file, it returns the hold file number as an error message number in the secondary input buffer.
Hold file numbers are returned as Entry #n, where "n" is the hold file number, allowing you to distinguish them from "real" error message numbers.

JCL COMMANDS

This section begins with a brief summary of the jCL commands, organized by a function after which follows a description in alphabetical order.

Input Buffer Commands

B	Moves the buffer pointer back to the previous parameter.
F	Moves the buffer pointer forward to the next parameter.
IBH	Inserts a text string containing embedded blanks into the active input buffer.
IH	Inserts a text string, creates a new null parameter, or nulls an existing parameter in the active input buffer.
RI	Use to clears all or part of the primary input buffer, and to clear the secondary input buffer.
S	Moves the input buffer pointer to a specified parameter or column.

Output Buffer Commands

BO	Moves the active output buffer pointer back one parameter.
H	Inserts a literal into the active output buffer.
RO	Clears both output buffers and selects the primary as active.
STOFF	Selects the primary as the active output buffer.
STON	Selects the secondary (stack) as the active output buffer.

Data Movement Commands

A	Copies a parameter from the active input buffer to the active output buffer.
MS	Move the secondary input buffer contents to the primary input buffer.
MV	Copies data between primary input buffer, active output buffer, file buffers and select registers.
MVA	Copies the specified source into the destination buffer and stores it as a multivalue.
MVD	Deletes data from a multivalued parameter in the destination buffer.

Input/Output Buffer Operations

IBN	Accepts input from the terminal as a single parameter with all blanks intact and places it in the secondary input buffer.
IBP	Accepts input from the terminal as a single parameter with all blanks intact and

	places it in the specified buffer or the active input buffer.
IN	Accepts input from the terminal and places it in the secondary input buffer.
IP	Accepts input from the terminal and places it in the specified buffer or the active input buffer.
IT	Transfers a tape record to the primary input buffer.

Jump and Branch Operations

( )	Terminates the current jCL program and begins execution of (chains to) another jCL program.
[]	Calls an external jCL program routine.
G, GO, GOTO	Transfers control to the jCL program statement with the specified label.
GO B	Transfers control backward to the previous M (mark) command and continue execution from that point.
GO F	Transfers control forward to the next M (mark) command and continue execution from that point.
GOSUB	Transfers control to the local subroutine with the specified label.
M	Marks a location to which a GO F or a GO B command transfers control.
RSUB	Terminates execution of the local subroutine and returns control to the statement following the GOSUB that called the subroutine.
RTN	Returns control from an external jCL program subroutine to the jCL program that called the subroutine.

Conditional Operations

IF	Allows conditional execution of jCL program commands.
IF E	Tests for presence of an error condition after processing a shell command.
IFN	Performs numeric comparisons and allows conditional execution of jCL program commands.

File Operations

F-CLEAR, F-C	Clears the specified file buffer.
F-DELETE, F-D	Deletes a record from a file opened by an F-OPEN command.
F-FREE, F-F	Releases a record lock set by the F-UREAD command.
F-KLOSE, F-K	Closes the specified file buffer.
F-OPEN, F-O	Clears and opens a file buffer to allow reads and writes.
F-READ, F-R	Reads a record from a file into a file buffer.
F-UREAD, F-UR	Reads a record from a file into a file buffer and locks the record.

F-WRITE, F-W	Writes the contents of the specified file buffer to a file.
FB	Reads a record into a special "fast buffer" without first opening the file.

Arithmetic Operators

+	Adds an integer to the current parameter in the active input buffer.
-	Subtracts an integer from the current parameter in the active input buffer.
F;	Performs arithmetic functions on constants and buffer parameters.

Processing

P	Executes the shell command in the primary output buffer.
PU	Executes the UNIX command in the primary output buffer, using the UNIX shell.

Terminal And Printer Output

L	Formats output to the printer.
O	Outputs a text string to the terminal.
T	Produces complex, formatted terminal output and display buffer values.

Debugging

C or *	Includes a comment in a jCL program.
D	Displays all or part of the active input buffer.
DEBUG	Turns debug on or off
TR	Before executing, it invokes a trace for a jCL program and displays each command on the terminal.
PP	Displays the command in the output buffer and prompts to continue.
PW	Displays the command in the output buffer and prompts to continue.

Exiting

( )	Terminates the current jCL program and begins execution of another jCL program.
X	Halts execution of a jCL program and returns control to the shell.

CHAPTER 2 JCL COMMAND SYNTAX

( )

Terminates the current jCL program and begins execution of (chains to) another jCL program.

SYNTAX

({DICT} file-name{, data-section-name} {key}) {label}

SYNTAX ELEMENTS

DICT specifies the dictionary section of file-name, if required.
file-name is the name of the file that contains the jCL program for execution. Can be a literal, or a direct or indirect reference to a buffer or select register.
data-section-name specifies an alternative data section of the file. Can be a literal, or a direct or indirect reference to a buffer or select register.
key is the name of the jCL program for execution. Can be a literal, or a direct or indirect reference to a buffer or select register. If no key is specified, it uses the current parameter in the active input buffer.
label specifies a label in the target jCL program from which to start execution.

NOTES

a. The ( ) command terminates the current jCL program and begins execution of another jCL program, of the same type.
b. Passed to the second program are the input buffers, output buffers, and file buffers, where all open files remain open.
c. Use the ( ) command in the MD to "point" to another jCL program, which contains the main body of code. See example 1.
d. Also, use the ( ) command to divide large jCL programs into smaller units, minimizing the search time for labels.
e. Using the ( ) command (or the [ ] command), will ensure that the contents of all buffers are available to the target program. If this is not a consideration, you can execute another jCL program with the P command (see later).

EXAMPLE 1

```
MENU
001 PQN
002 (JCL MENU2)
```

Immediately executes another jCL program called MENU2 in the file called JCL.

EXAMPLE 2

```
MENU
001 PQN
002 (JCLFILE)
```

When entering the word MENU from the shell, place it in parameter 1 of the primary input buffer - %1. The program will then unconditionally pass control to the MENU program in file JCLFILE.

EXAMPLE 3

```
DOIT
001 PQ
002 IP?
003 (JCL)
```

This example will prompt for the name of another jCL program and continue execution with the named jCL program in the file called JCL.

EXAMPLE 4

```
MENU
001 PQN
002 (JCL MENU2) 300
```

Immediately executes the jCL program called MENU2 in the file called JCL. Execution of MENU2 will begin at label 300

[]

Calls another jCL program as an external subroutine

SYNTAX

[{DICT} file-name{, data-section-name} {key}] {label}

SYNTAX ELEMENTS

DICT specifies the dictionary level of file-name, if required.
file-name the name of the file, which contains the jCL program subroutine. Can be a literal or a direct or indirect reference to a buffer or select register.
data-section-name specifies an alternative data section of the file (default is the same name as the dictionary). Can be a literal or a direct or indirect reference to a buffer or select register.
key is the name of the jCL program for execution. Can be a literal or a direct or indirect reference to a buffer or select register. If no key is specified it uses the current parameter in the active input buffer.
label specifies a label in the target jCL program from which to start execution. Use of the label clause makes this command synonymous with the GOSUB command.

NOTES

a. Passed through to the called program are the input buffers, output buffers, and file buffers where all open files remain open.
b. External subroutines can call other subroutines.
c. You can make an unlimited number of calls but the jCL programs must be of the same type.
d. When encountering an RTN, it returns control to the calling jCL program; encountering no RTN, terminates execution at the end of the called program.

EXAMPLE

```
001 PQN
002 [SUBS SUB1]
003 ...
```

Calls the jCL program SUB1 in the SUBS file as an external subroutine.

+

Adds an integer value to the current parameter in the active input buffer

COMMAND SYNTAX

+n

SYNTAX ELEMENTS

n is the integer to add.

NOTES

a. If the number of characters in a parameter increases because of the + command, it moves the remaining parameters to the right.

b. If a parameter is preceded by a + sign, it replaces the sign with a zero.

c. If the buffer pointer is at the end of the buffer, it creates a new parameter.

d. If the referenced parameter is non-numeric, it uses a zero.

EXAMPLE 1

Command	PIB Before	PIB After
+30	AAA^+20^333 a	AAA^050^333 a

EXAMPLE 2

Command	PIB Before	PIB After
+100	BBB^AA^44 a	BBB^100^44 a

EXAMPLE 3

Command	PIB Before	PIB After
+51	ABC^0000^55 a	ABC^0051^55 a

-

Subtracts an integer from the current parameter in the active input buffer

SYNTAX

-n

SYNTAX ELEMENTS

n is the integer to subtract.

NOTES

a. If the number of characters within a parameter decreases with a - command, it prefixes the result with zeros to maintain the same number of characters as the original value.

b. A minus sign can precede parameters within the input buffer. If the buffer pointer is at the end of the buffer, it creates a new parameter. If the referenced parameter is non-numeric, it uses a zero.

EXAMPLE 1

Command	PIB Before	PIB After
-300	AAA^345^666 A	AAA^045^666 a

EXAMPLE 2

Command	PIB Before	PIB After
-20	AAA^ABC^666 a	AAA^-20^666 a

EXAMPLE 3

Command	PIB Before	PIB After
-50	AAA^-50^666 A	AAA^-100^666 a

EXAMPLE 4

```
001 PQN
002 OEnter a number+
003 S5
004 IBP
005 +7
006 T %5
007 –3
008 T %5
009 RTN
```

This example receives input from the terminal and places it in the 5th parameter of the primary input buffer. It adds 7 to the value stored in the 5th parameter and displays the result. It then subtracts 3 from the result and displays the new value.

CHAPTER 3 JCL STATEMENTS

A	Copies a parameter from the active input buffer to the active output buffer.
B	Moves the active input buffer pointer back to the previous parameter.
BO	Moves the active output buffer pointer back by one parameter.
C	Defines a comment.
D	Displays parameters from the active input buffer.
DE	Displays the current value of LastError.
DEBUG	Turns the jCL debug function on or off.
F	Moves the active input buffer pointer forward to the next parameter.
F;	Provides a range of arithmetic functions.
F-CLEAR	Clears the specified buffer.
F-DELETE	Deletes a record from a file.
F-FREE	Releases a record lock.
F-KLOSE	Closes a specified file buffer.
F-OPEN	Opens a file for reading and writing.
F-READ	Reads a record from an open file into a file buffer.
F-UREAD	Reads and locks a record from an open file into a file buffer.
F-WRITE	Writes the contents of a file buffer as a record.
FB	Reads a record from a file into the special "fast" buffer without having to open the file first.
G/GO/GOTO	Transfers control unconditionally to another location in the program.
GO B	Transfers control to the statement following the most recent mark command executed.
GO F	Transfers control to the statement containing the next mark command.
GOSUB	Transfers control to a specific subroutine.
H	Places a text string in the active output buffer.
IBH	Places text in the active input buffer whilst retaining embedded spaces.
IBN	Prompts for input and places the entered data in the secondary input buffer.
IBP	Prompts for input from the terminal.
IF	Allows conditional execution of jCL commands based on the evaluation of an expression.
IF E	Conditionally executes a command depending on the presence or absence of an error message.
IF S	Conditionally executes a command depending on the presence or absence of an active select list.

IFN	Allows conditional execution of commands depending on the result of numeric comparisons.
IH	Places a text string in the active input buffer.
IN	Prompts for input and places it in the secondary input buffer.
IP	Prompts for input and places it into the active input buffer or a nominated buffer.
IT	Reads a tape record into the primary input buffer.
L	Formats printed output.
M	Marks a destination for a GO F or GO B command
MS	Move the entire content of the secondary input buffer to the primary input buffer.
MV	Copies data between buffers or between buffers and select registers.
MVA	Copies a value from the source to the destination buffer and stores it as a multivalue.
MVD	Deletes a value from a multivalued parameter in the target buffer.
O	Outputs a text string to the terminal.
P	Submits the shell command created in the primary output buffer for execution.
PQ-RESELECT	Executed from a jCL program, resets the pointer of a select register to the beginning of the list of keys.
PQ-SELECT	Executed from a jCL program, loads a list of keys into a select register
RI	Resets (clears) the primary and secondary input buffers.
RO	Resets (clears) the active output buffer.
RSUB	Terminates execution of a local subroutine.
RTN	Terminates execution of an external subroutine.
S	Positions the primary input buffer pointer to a specified parameter or column.
STOFF	Selects the primary output buffer as the active output buffer.
STON	Selects the secondary output buffer as the active output buffer.
T	Produces formatted terminal output.
TR	Before executing, it traces jCL program execution and displays each command.
U	Executes a user exit from a jCL program.
X	Halts execution of the program and returns control to the shell.

PQ and PQN Differences

- The first line of a jCL program defines the program as one of two basic types, a PQ or a PQN style program.
- Wherever possible, you should use the PQN style.

There are several differences between the two styles, as outlined in the following topics.

DELIMITERS

- PQ-style programs usually use a ‘space’ as the delimiter between parameters in the buffers. PQN-style programs usually use a field mark.
- PQN allows parameters to be null or contain blanks and more closely mirrors the native record structure.
- Still supports PQ commands for compatibility but you should use the functionally superior PQN commands in new or revised jCL programs.
- When moving pointers, PQ commands will generally leave the pointer at the first character of a parameter. PQN commands will generally leave the pointer at a field mark.
- Commands affected by this difference are A, B, BO, F, H, IH and IP.

BUFFERS

Buffer referencing, file buffers and select registers are only available with PQN commands.

COMMANDS

PQN-style jCL programs use only these commands:

F;	F-KLOSE,	F-WRITE,	L,	MVD
F-CLEAR	F-OPEN	FB	MS	
F-DELETE	F-READ	IBH	MV	
F-FREE,	F-UREAD	IBP	MVA	

These commands are functionally equivalent in both PQ and PQN-style programs:

( )	G,	IF E,	RSUB,	U
[],	GO B,	IFN,	RTN,	X
+,	GO F,	M,	S,	
-,	GOSUB,	P,	STOFF,	
C,	IF,	RI,	STON,	

JCL PQ-SELECT

Executed from a jCL program, loads a list of keys into a select register

COMMAND SYNTAX

PQ-SELECT register-number

SYNTAX ELEMENTS

register number is the number of the select register (1 to 5) in which to place the keys.

NOTES

a. To use PQ-SELECT you must first construct a list by using one of the list processing commands such as SELECT, SSELECT, QSELECT, BSELECT, GET-LIST, FORM-LIST, SEARCH or ESEARCH.

b. Put the PQ-SELECT command in the stack for processing as part of the external job stream when the required list is active.

c. Use a "!n" direct or an indirect reference to retrieve the list elements one at a time.

d. You cannot execute PQ-SELECT directly from the shell.

EXAMPLE

```
001 PQN
002 HSSELECT SALES
003 STON
004 HPQ-SELECT 1
005 P
006 10 MV %1 !1
007 IF # %1 X Done
008 T %1
009 GO 10
```

This example selects all the records in the SALES file, loads the record-list into select register 1 and displays the keys one at a time.

JCL PQ-RESELECT

Executed from a jCL program, it resets the pointer of a specified select register to the beginning of the list of record keys.

COMMAND SYNTAX

PQ-RESELECT register-number

SYNTAX ELEMENTS

register-number is the number (1 to 5) of the select register for reset.

NOTES

a. Executed from the primary output buffer this command resets the pointer of a specified select register back to the beginning of the list.
b. Each time you use the "!" reference to retrieve a value from the list the value is not destroyed it simply advances the pointer to the next parameter in the list.
c. PQ-RESELECT resets the pointer to the beginning of the list to allow you to perform another pass.
d. You cannot execute PQ-RESELECT directly from the shell.

EXAMPLE

```
HSELECT SALES
STON
HPQ-SELECT 1
PH
MV %1 !1
IF # %1 XNo records selected
HPQ-RESELECT 1
PH
10 MV %1 !1
IF # %1 XFinished
...
GO 10
```

CHAPTER 4 JCL COMMANDS

JCL A

Copies a parameter from the active input buffer to the active output buffer

COMMAND SYNTAX

A{c|\}{p}
A{c|\}{p}({n},m)

SYNTAX ELEMENTS

C specifies a character to surround the string being copied. Can be any non-numeric character except left parenthesis "(" or backslash "\".
\ specifies that the value be concatenated with the current parameter in the active output buffer.
p specifies the number of the parameter to be copied. If p is not specified, it copies the current parameter.
n specifies the starting column in the PIB. Copying continues until it reaches the end of the parameter or has copied the number of characters specified by m.

NOTES

a. Used on its own, the A command will copy the parameter pointed to from the active input buffer to the active output buffer.
b. Use the A command with the IF command
c. If the active input buffer pointer is at the end of the buffer, the A command will have no effect.
d. Use the c option when you need to surround keys with single quotes, or values with double quotes.
e. If you include an **n** or **m** specification, it selects the PIB. If the destination is the SOB (stack is "on"), it ignores c.
f. If the stack is "off" (the POB is active), the A command will place the data in the output buffer as a separate parameter. It positions the buffer pointers in the primary output buffer at the field mark at the end of the copied parameter.
g. If the stack is "on" (the SOB is active), the A command will concatenate the value to the previous parameter in the output buffer. It will continue to copy until encountering a field

mark. When complete, it positions the buffer pointers at the end of the secondary output buffer.

EXAMPLE 1

Command	PIB Before	PIB After
A	AAA SALES^JAN ^	AAA^SALES^JAN ^
	POB Before	**POB After**
	LIST^ ^	LIST^SALES^ ^

NOTE: the position of the buffer pointer after you issue the A command

EXAMPLE 2

Command	PIB Before	PIB After
A	AAA^SALES^JAN ^	AAA^SALES^JAN ^
	POB Before	**POB After**
	LIST^SALES^ ^	LIST^SALES^JAN ^ ^

Issuing an A"3 command would have achieved the same result, except that the starting position of the PIB pointer would have been immaterial.

EXAMPLE 3

Command	PIB Before	PIB After
A\	ABC^DEF^GHI ^	ABC^DEF^GHI ^ ^
	POB Before	**POB After**
	XXX^ ^	XXXABC^ ^

EXAMPLE 4

Command	PIB Before	PIB After
A2 (2,-2)	ABC^MYLIST^JKL ^	ABC^MYLIST^JKL ^
	POB Before	**POB After**
	SAVE-LIST^ ^	SAVE-LIST MYLIST ^

The command attempts to copy the second parameter from the PIB, starting with the second character, up to and including, the penultimate character, to the current output buffer.

JCL B

Moves the active input buffer pointer back to the previous parameter.

COMMAND SYNTAX

B

NOTES

Moves the input buffer pointer backwards to the preceding field mark or to the beginning of the input buffer. If the pointer is on the first character of a parameter (or a field marker), it moves the pointer back to the field mark of the previous parameter.

EXAMPLE 1

Command	PIB Before	PIB After
B	ABC^DEF^GHIJK ^	ABC^DEF^GHIJK ^

EXAMPLE 2

Command	PIB Before	PIB After
B	ABC^DEF^GHIJK ^	ABC^DEF^GHIJK ^

JCL BO

Moves the active output buffer pointer back by one parameter.

COMMAND SYNTAX

BO

NOTES

The buffer pointer will move backward until it finds a field mark or the start of the buffer.
To completely clear the buffer, use the RO command. To clear specific parameters, use the MV #n command

EXAMPLE 1

Command	POB Before	POB After
BO	ABC^DEF^GHIJK ^	ABC^DEF^GHIJK ^

EXAMPLE 2

Command	SOB Before	SOB After
BO	SAVE-LIST ^	SAVE-LIST ^

JCL D

Displays the current parameter of the active input buffer or specific parameters from the PIB

COMMAND SYNTAX

D{n}{+}

SYNTAX ELEMENTS

n specifies the number of the PIB parameter for display. If setting **n** to 0 (zero), it displays all parameters in the primary input buffer.

+ inhibits a NEWLINE at the end of output.

NOTES

- D with no other qualifiers will display the current parameter of the active input buffer. It will not change the pointer position.

EXAMPLE 1

Command	Active Input Buffer	Display
D	ABC^DEF^GHI	DEF

EXAMPLE 2

Command	Active Input Buffer	Display
D3	ABC^DEF^GHI	GHI

EXAMPLE 3

Command	Active Input Buffer	Display
D0	ABC^DEF^GHI	ABC^DEF^GHI

JCL D

Displays the current parameter of the active input buffer or specific parameters from the PIB

COMMAND SYNTAX

D{n}{+}

SYNTAX ELEMENTS

n specifies the number of the PIB parameter for display. If setting **n** to 0 (zero), it displays all parameters in the primary input buffer.
+ inhibits a NEWLINE at the end of output.

NOTES

D with no other qualifiers will display the current parameter of the active input buffer. It will not change the pointer position.

EXAMPLE 1

Command	Active Input Buffer	Display
D	ABC^DEF^GHI ^	DEF

EXAMPLE 2

Command	Active Input Buffer	Display
D3	ABC^DEF^GHI ^	GHI

EXAMPLE 3

Command	Active Input Buffer	Display
D0	ABC^DEF^GHI	ABC^DEF^GHI

JCL DE

Displays the current value of LastError

COMMAND SYNTAX

DE

NOTES

Use the DE command when debugging your jCL programs.

EXAMPLE

If LastError contains 404]61^QLNUMSEL, the DE command will display: 404]61^QLNUMSEL

JCL DEBUG

Turns debug function on or off.

COMMAND SYNTAX

DEBUG [ON|OFF]

NOTES

When turning the DEBUG function on, it suspends your jCL program before executing each command (line). You will see the program name and the next command for execution; at the prompt enter one of the following commands:

Command	Description
<Enter>	Execute current line
?	Display list of available commands
E	Toggle last error display
f	toggle file buffer display
h	display list of available commands
i	toggle input buffer display
n	toggle next line display between one and two lines
o	toggle output buffer display
q	quit (terminate) program
x	exit DEBUG function

After each input, it displays the program name and the next line for execution, together with any additional information requested (for example, the content of the input buffers).

JCL F

Moves the active input buffer pointer forward to the next parameter.

COMMAND SYNTAX

F

NOTES

Moves the input buffer pointer forward to the next field mark, or to the end of the buffer.

EXAMPLE 1

Command	PIB Before	PIB After
F	ABC^DEF^GHI ^	ABC^DEF^GHI ^

EXAMPLE 2

Command	PIB Before	PIB After
F	ABC^DEF^GHI ^	ABC^DEF^GHI ^

JCL. F

Provides a range of arithmetic functions

COMMAND SYNTAX

F;element{;element}...;?[P|r]

SYNTAX ELEMENTS

Element operators or operands (see below)

?P moves the result (top stack entry) into the current parameter of the primary input buffer.

?r moves the result (top stack entry) into the reference r. Can be a direct or indirect reference to a buffer.

OPERATORS

+	Add last stack entry and penultimate stack entry. Store the result in stack entry 1.
-	Subtract last stack entry from penultimate stack entry. Store the result in stack entry 1.
*	Multiply the last two stack entries. Store the result in stack entry 1.
/	Divide stack entry 2 by stack entry 1. Store the result in stack entry 1.
R	Divide stack entry 2 by stack entry 1. Store the remainder in stack entry 1.
_	Reverse the last two stack entries.

OPERANDS

r	A direct or indirect reference to a buffer or select register value for placing on the stack.
{C}literal	Literal value specified by *literal*.

NOTES

a. The F; command uses a stack processor similar to the one used by the F; function in jQL.
b. Processes commands from left to right.
c. Each operand pushes a value on to the top of the push-down/pop-up stack.
d. Copy the result of a calculation into any buffer with the question mark (?) operator.

EXAMPLE

Command	PIB Before	PIB After
F;C100;%2;-;?%3	6^8^999^7^GHI	6^8^92^7^GHI
Stack	100	

%2 pushes a value of 8 (the 2nd parameter of the PIB) onto the stack.

```
Stack               100
                    8
```

- subtracts last entry from penultimate entry, and replaces last entry 1 with the result of 92

```
Stack               92
```

?%3 then copies the result to the 3rd parameter of the PIB, replacing the old value.

JCL F-CLEAR

Clears the specified file buffer

COMMAND SYNTAX

F-CLEAR file-buffer

F-C file-buffer

SYNTAX ELEMENTS

file-buffer is the number (1 to 9) of the file buffer for clearing.

NOTES

This command is equivalent to using the MV file-buffer.0 ",_ command

EXAMPLE

```
001 PQN
002 F-C 1
003 MV &1.0 "Key", "Field 1"
```

Clear file buffer 1. Then set the record key to "Key" and the first field to "Field 1". Functionally equivalent to MV &1.0 "Key", " Field1",_. (Note the use of the underscore character as the last character of the command).

JCL F-DELETE

Deletes a record from a file

COMMAND SYNTAX

F-DELETE file-buffer

F-D file-buffer

SYNTAX ELEMENTS

file-buffer is the number (1 to 9) of the file buffer, which is associated with the file containing the record for deletion.

NOTES

a. Deletes the record specified in field zero (&f.0) of the file buffer. If the record does not exist, the command will have no effect.
b. Does not alter the content of the file buffer.
c. Will release an outstanding record lock.
d. You must open the file an F-OPEN Command.

EXAMPLE

```
001 PQN
002 10 T "File name :",+
003 IBP %1
004 F-O 1 %1
005 T "File ", %1, " does not exist!"\ GO 10
006 MV &1.0 "Key"
007 F-D 1
```

If the F-O command cannot open the file, it executes the line immediately following the command (see the F-O command). If opened, it moves "Key" into field 0 of file buffer 1. F-D 1 then attempts to delete record "Key" from the file.

JCL F-FREE

Releases a record lock set by the F-UREAD command.

COMMAND SYNTAX

F-FREE {file-buffer {key}}
F-F {file-buffer {key}}

SYNTAX ELEMENTS

file-buffer specifies a file buffer (1 to 9) assigned by an F-OPEN command.
key is the key of the record to be unlocked. Can be a literal (not enclosed in quotes), or a direct or indirect reference to a buffer or select register.

NOTES

a. If file-buffer is not specified, all record locks set by the jCL program on the current port, are released within the current context level.
b. If no key is specified, it releases any outstanding locks for the current file.
c. Record locks are also released by F-WRITE or F-DELETE commands, and at the end of the jCL program. Use the LIST-LOCKS command to see which records are currently locked.

EXAMPLE 1

F-FREE
Unlocks all records previously locked by this jCL program.

EXAMPLE 2

F-FREE 1
Unlocks the record specified by the key in field zero of file buffer 1.

EXAMPLE 3

F-F 1 Key
Unlocks the record "Key" in the file associated with file buffer 1.

JCL FKLOSE

Closes a specified file buffer

COMMAND SYNTAX

F-KLOSE file-buffer

FK file-buffer

SYNTAX ELEMENTS

file-buffer is the number (1 to 9) of the file buffer to be closed.

NOTES

Use F-KLOSE when you have finished working with a file you need to close.

EXAMPLE

F-K 1

Closes file buffer 1.

JCL F-OPEN

Clears a file buffer and opens a file for reading and writing.

COMMAND SYNTAX

F-OPEN file-buffer {DICT} file-name{,data-section-name}
error-cmd-line
F-O file-buffer {DICT} file-name{,data-section-name}
error-cmd-line

SYNTAX ELEMENTS

file-buffer is the number (1 to 9) of the file buffer with which the file is to be associated.
DICT specifies the dictionary section of file-name, if required.
file-name is the name of the file to be opened. Can be a literal (not enclosed in quotes), or a direct or indirect reference to a buffer or select register.
data-section-name specifies an alternative data section of file-name.
error-cmd-line is the line immediately after the F-OPEN command and it executes only if it cannot open the specified file.

NOTES

a. If the file cannot be opened, it executes the line immediately after the F-OPEN command. If the file is opened successfully, it will ignore this line.
b. File buffers are maintained when control is transferred between jCL programs.
c. The file will remain open until closed (see the F-KLOSE command) or until the end of the program.

EXAMPLE

```
001 PQN
002 F-OPEN 1 SALES
003 X ERROR: Can’t find the Sales File!
004 T C, (5,10), "Welcome to...",+
```

If it opens the SALES file, execution continues with line 004, else, the program terminates with an appropriate error message.

JCL F-READ

Reads a record from an open file into a file buffer.

COMMAND SYNTAX

F-READ *file-buffer key*

error-cmd-line

F-R *file-buffer key*

error-cmd-line

SYNTAX ELEMENTS

file-buffer is the number (1 to 9) of the file buffer with which the file is associated.
key is the key of the record for reading. Can be a literal (not enclosed in quotes), or a direct or indirect reference to a buffer or select register.
error-cmd-line is the line immediately after the F-READ command, which executes only if it cannot read the specified record.

NOTES

a. If an associated file has not been opened (see the F-OPEN command), the program terminates with an error message.
b. If the specified record is not on file, it executes the line immediately following the F-READ command. If the read is successful, it ignores this line.

EXAMPLE

```
001 PQN
002 F-OPEN 1 SALES
003 X ERROR: Can't find the Sales File!
004 T C, (5, 10), "Welcome to...",+...
015 F-READ 1 ABC
016 X ERROR: Record ABC not found!
017 T "Record ABC found"
```

Attempts to read record **ABC** from the SALES file into file buffer 1. If not found, the program terminates with an appropriate error message, if found, it displays the message on line 17 and execution continues.

JCL F-UREAD

Reads and locks a record from an open file into a file buffer.

COMMAND SYNTAX

F-UREAD file-buffer key error-cmd-line
F-UR file-buffer key error-cmd-line

SYNTAX ELEMENTS

file-buffer is the number (1 to 9) of the file buffer with which the file is associated.
key is the key of the record for reading and locking. Can be a literal (not enclosed in quotes), or a direct or indirect reference to a buffer or select register.
error-cmd-line is the line immediately after the F-UREAD command; only executed if it cannot read the specified record.

NOTES

a. The F-UREAD command is identical to the F-READ command, except that it also locks the record against access by another process, thus eliminating simultaneous updates.
b. If you attempt to F-UREAD a locked record, it suspends execution until the other process unlocks the record.
c. F-DELETE, F-WRITE or F-FREE commands release the record locks, or when the program terminates.
d. It is good practice to F-UREAD a record before you create it. This will reserve the key in the file and avoid double updates. Remember: that the command line immediately following the F-UREAD command will be executed because the record does not exist.

JCL F-WRITE

Writes the contents of a file buffer as a record.

COMMAND SYNTAX

F-WRITE file-buffer

F-W file-buffer

SYNTAX ELEMENTS

file-buffer is the number (1 to 9) of the file buffer with which the target file is associated.

NOTES

a. The key of the record is contained in field zero of the file buffer; will not write the record if this field is null.
b. F-WRITE will not alter the contents of the file buffer.
c. You should not normally attempt to write a record unless you have first locked it with an F-UREAD command; releases the lock when the F-WRITE is complete.
d. The program will terminate with an error message if it cannot open the file (see the F-OPEN command).

EXAMPLE

```
001 PQN
002 F-OPEN 1 SALES
003 X ERROR: Can't find the Sales File!
004 T C, (5, 10), "Welcome to...",+
015 F-UREAD 1 ABC
016 F-F 1 \ G 1002
017 T "Record ABC found"
018 MV &1.2 "Field 2"
019 F-WRITE 1
. . .
. . .
999 1002 X Free Failed
```

.

Line 15 reads and locks record ABC in file SALES. If the record does not exist, it releases the lock on line 16 and transfers control to label 1002. If read successfully, it overwrites field 2 on line 18, where it writes the record back to the file on line 19 and unlocks it.

JCL FB

Reads a record from a file into the special "fast" buffer without having to open the file first.

COMMAND SYNTAX

FB {DICT} file-name{,data-section-name} {key}
error-cmd-line
FB ({DICT} file-name{,data-section-name} {key})
error-cmd-line

SYNTAX ELEMENTS

DICT specifies the dictionary section of file-name, if required.
file-name is the name of the file for opening. Can be a literal (not enclosed in quotes), or a direct or indirect reference to a buffer or select register.
data-section-name specifies an alternative data section of file-name.
key is the key of the record for reading. Can be a literal (not enclosed in quotes), or a direct or indirect reference to a buffer or select register. If key is not specified, it uses the value at the active input buffer
pointer to supply the key
error-cmd-line is the line immediately after the FB command; executes only if it cannot open the specified file or read the record.

NOTES

a. Each time you use the FB command, it overwrites the previous contents of the buffer.
b. The FB command is useful if you are only reading one record from a file. Otherwise, you should use the F-OPEN and F-READ commands.
c. If the specified record is not on file, or if the file does not exist, it executes the line immediately following the FB command. If the read is successful, it ignores this line.
d. Subsequent references to the fast buffer use a special syntax. For example, to refer to the second field of a record in the fast buffer, you would use &2.

EXAMPLE 1

```
001 PQN
002 FB SALES ABC
003 T "ABC not on file" / G 1001
004 MV %3 &2
```

The FB command on line 2 attempts to read record ABC from file SALES. If not found for any

reason, it outputs a message and transfers control to label 1001 by line 3. If read successfully, execution continues at line 004, which moves field two of the record into parameter 3 of the PIB.

EXAMPLE 2

```
001 PQN
002 T C, (5, 10), "Name :",+
003 IP %2
004 FB SALES %2
005 T "New record"
006 T "Area :",+
007 IP %3
```

Prompts the user for a name (the record key) and uses the fast buffer to read the record from the SALES file. If the record does not exist, it outputs a message a message but continues processing.

JCL GO B

Transfers control to the statement following the most recent M (mark) command executed.

COMMAND SYNTAX

G B

GOB

GOTO B

NOTES

GO B returns to the last executed M, no matter where located in the program.

If a "mark" has not previously been executed, the program terminates with an error message:

Can't find mark at line n in program name

EXAMPLE 1

```
001 PQN.
010 MV #1 "SSELECT SALES BY MONTH"
011 STON
012 MV #1 "PQ-SELECT 1"
013 P
014 M
015 MV %1 !1
016 IF # %1 GO F
017 C Process the record
025 GO B
026 M
```

Lines 10 to 13 create a sorted list of the records in the SALES file. After each record is processed, the GO B command on line 25 returns control to the M command on line 14. When the end of the list is reached, the IF command on line 16 transfers control to the M command on line 26.

EXAMPLE 2

```
001 PQN
009 MV %1 ","
010 M
011 IF # %1 GO 30
012 M
013 IF # %2 GO 40
014 GO 50
015 30 MV %1 "ABC"
016 GO B
017 40 MV %2 "DEF"
018 GO B
019 50 ....
```

This example simply serves to demonstrate how the GO B command will remember the last M command rather than search backwards for it. First, the values in %1 and %2 are set to null in line 9 and records the M at line 10. When control reaches line 11, it tests %1 and transfers control to label 30 on line 15. It does not record the intervening M command (at line 12). Line 15 assigns a value of ABC to %1. Line 16 returns control to the M on line 10. %1 does have a value; control moves on to line 12 and records the M on this line. Next, it tests %2 at line 13 and transfers control to label 40 on line 17. When processing the GO B on line 18, it transfers control back to line 12.

JCL GO F

Transfers control to the statement containing the next M (mark) command.

COMMAND SYNTAX

G F
GO F
GOTO F

NOTES

a. Scans the program forward from the current line, until it locates the next M command. Program execution then jumps to that line.
b. If it cannot locate an M command cannot, the jCL program will terminate with an error message:

Can't find mark at line n in program name

EXAMPLE

```
001 PQN
005 GO F
006 10 MV %1 "ABC"
007 MV %6 "DEF"
008 MV %10 "HIJ"
009 M
```

.
The GO F command on line 5 causes the program to be scanned from line 6, looking for the next M command. In this case, it transfers control to line nine.

JCL GOSUB

Transfers control to a specific subroutine.

COMMAND SYNTAX

GOSUB label

GOSUB label] label... (Multivalued form)

SYNTAX ELEMENTS

label specifies the label, which marks the required subroutine.

NOTES

When encountering an RSUB statement, it transfers control back to the line immediately following the calling GOSUB.
See also "[ ] {n}" command.

MULTIVALUED FORM

To use the multivalued form of the GOSUB command, you must specify one label for each result of a multiple comparison. For example:
IF %2 = A]B]C]D GOSUB 1000]2000]3000]4000
Separate the test values and the destination labels with value marks (ctrl]).

NOTE: that this is a special case of the GOSUB command. If you need to mix command types in the resulting actions, you should not use this form of the GOSUB command. You can still achieve the same effect but each result must contain a separate command.

EXAMPLE:

IF %2 = A]B]" GOSUB 1000]GOSUB 2000]XFinished
In this case, if the result of the test for null is true the program will terminate with a suitable message.

EXAMPLE

```
010 GOSUB 1000
011 T "Returned from GOSUB"
031 1000 T "In subroutine"
032 IP %1
033 RSUB
```

The GOSUB command on line 10 transfers control to label 1000 on line 31 When the RSUB on line 33 is processed, control returns to line 11.

JCL G / GO / GOTO

Transfers control unconditionally to another location in the program.

COMMAND SYNTAX

G label

GO label

GOTO label

GO label] label... (Multivalued form used with multivalued IF command)

SYNTAX ELEMENTS

label specifies the location from which execution is to continue.

NOTES

a. If the label has not been encountered in the program, GOTO will search for the label, from the current position. The target label must be found at the beginning of a command line, separated from the command by at least one space.

b. If the label cannot be found, or is defined more than once, the program will terminate with an error message.

MULTIVALUED FORM

To use the multivalued form of the GO command, you must specify one label for each result of a multiple comparison. For example:

```
IF %2 = A]B]C]D GO 10]20]30]40
```

Separates the test values and the destination labels with value marks (ctrl]).

NOTE: that this is a special case of the GO command. If you need to mix command types in the resulting actions, you should not use this form of the GO command. You can still achieve the same effect but each value must contain a separate command. For example:

```
IF %2 = A]B]C]" GO 10]GO 20]GO 30]XFinished
```

In this case, if the result of the test for null is true the program will terminate with a suitable message.

EXAMPLE 1

```
001 PQN
002 F-OPEN 1 SALES
003 G 1001
004 T C, (5, 10), "Welcome to...",+
```

```
087 1001 T "ERROR: Can"t find the Sales File!"
088 IP %99
089 RTN
```

.
If it opens the SALES file, execution continues with line 004. Otherwise, it transfers control to label 1001 on line 87.

EXAMPLE 2

```
022 5 T "Option :",+
023 IP %1
024 IF %1 = A]B]C GO 10]20]30
025 GOTO 5
```

.
This example transfers control

- To label 10 if entering "A",
- To label 20 if entering "B" is entered
- To label 30 if entering "C".

If the response is not recognized, it controls transfers back to label five.

JCL H

Places a text string in the active output buffer

COMMAND SYNTAX

Htext-string

SYNTAX ELEMENTS

text-string is the text for placing in the active output buffer. Can be a literal (not enclosed in quotes), or a direct or indirect reference to a buffer or select register.

NOTES

a. Use the H command to place a text string in the currently active output buffer.
b. Use the POB, to create a shell command.
c. Use the SOB to create secondary commands (such as PQ-SELECT) or to "stack" a series of inputs required by the active process.
d. Moves the string into the buffer starting at the current location of the buffer pointer. At the end of the operation, it positions the buffer pointer immediately after the last character in the string.
e. If quotes are used to delimit the string everything within quotes will be treated as a single field and the string will be moved into the buffer as a single parameter.
f. If quotes are not used, it replaces each group of one or more spaces in the string by a field mark as it moves the text into the buffer. Include a leading space if you want to add a new parameter. If you want to concatenate the string to the current buffer parameter, do not use a leading space.
g. Use the P command to process the contents of the POB and SOB.

USING H WITH THE PRIMARY OUTPUT BUFFER

When issuing the shell command, it replaces the field marks by spaces and automatically appends a carriage return.

USING H WITH THE SECONDARY OUTPUT BUFFER

Does not append a carriage return automatically to output from the SOB; Terminate each line with a less than character (<) to represent a carriage return.

EXAMPLE 1

```
001 PQN
002 HSLEEP 10
003 P
```

Forces the process to sleep for 10 seconds

EXAMPLE 2

Command	POB Before	POB After
HCOPY		COPY
H SALES	COPY	COPY^SALES
H ABC	COPY^SALES	COPY^SALES^ABC
H-DEF	COPY^SALES^ABC	COPY^SALES^ABC-DEF
H (P)	COPY^SALES^ABC-DEF	COPY ^SALES^ABC-DEF^(P)

NOTE: how COPY and SALES have become separate parameters but ABC and -DEF have been concatenated

EXAMPLE 3

```
001 PQN
002 HGET-LIST LISTA
003 STON
004 HCOPY SALES<
005 H(SALES.HOLD
006 P
```

JCL IBH

Places text in the active input buffer whilst retaining embedded spaces and applying any required input/output conversions.

COMMAND SYNTAX

IBHtext
IBHreference;input-conversion;
IBHreference:output-conversion:

SYNTAX ELEMENTS

text placed in the active input buffer. Can be a literal (not enclosed in quotes), or a direct or indirect reference to a buffer or select register.
reference is a direct or indirect reference to a buffer or select register.
input-conversion is a jQL input conversion for applying to the string before putting it in the buffer.
output-conversion is a jQL output conversion for applying to the string before putting it in the buffer.

NOTES

a. The IBH command works in the same way as the IH command except moving the string is as a single parameter, which maintains all spaces.
b. Depending on the position of the buffer pointer, IBH either will replace an existing parameter or adds a new parameter to the end of the input buffer. The rules are as follows:

    i. If the buffer pointer is at the beginning of an existing parameter, it replaces that parameter with the new string.
    ii. If the buffer pointer is within an existing parameter, IBH will concatenate the new string (without inserting a field mark), starting at the current location of the buffer pointer.
    iii. If the buffer pointer is at the end of the input buffer, it creates a new parameter leaving the buffer pointer will pointing to the field mark preceding the new parameter.
    iv. In all cases, the position of the buffer pointer will remain unchanged.
    v. Conversions containing colons or semicolons will not work (for example IBH;G1;1;).

EXAMPLE 1

Command	PIB Before	PIB After
IBHDEF GHI	ABC^XXX^Z ^	ABC^DEF GHI^Z

EXAMPLE 2

Command	PIB Before	PIB After
IBH XX	AA^BB^CC^DD ^	AA^BB^CC^DD^ XX

EXAMPLE 3

File buffer 1 contains:

```
000 Key
001 11350
```

Command	PIB Before	PIB After
IBH&1.1:D2:	AA^BB^CC^DD ^	AA^BB^27 JAN 99^DD

JCL IBN

Prompts for input, places the entered data in the secondary input buffer as a single parameter and maintains embedded spaces. The secondary input buffer becomes as the active input buffer.

COMMAND SYNTAX

IBN{c}

SYNTAX ELEMENTS

c is an optional prompt character, which, once used, remains in effect until it issues a new IBN, IBP, IN or IP command. If not specified, the prompt character will default to the last prompt character used, or to a colon (:).

NOTES

a. The IBN command is similar to the IN command except that the input string is placed in the buffer as a single parameter and all spaces are maintained.
b. The new data replaces the content of the secondary input buffer, and the secondary input buffer will remain active until it uses an RI, S(n) or MV %n source command.
c. If the user responds with ENTER only, it creates a null parameter.

EXAMPLE 1

Input	SIB Before	SIB After
ABC	XXX ^	ABC

EXAMPLE 2

Input	SIB Before	SIB After
ABC DEF	XXX ^	ABC DEF

EXAMPLE 3

Input	SIB Before	SIB After
<ENTER>	XXX ^	

JCL IBP

Prompts for input from the terminal Input; holds data as a single parameter and retains embedded spaces.

COMMAND SYNTAX

IBP{c{r}}

SYNTAX ELEMENTS

c is an optional prompt character, which, once used, remains in effect until it issues a new IBN, IBP, IN or IP command. If not specified, the prompt character will default to the last prompt character used, or to a colon (:).

r is a direct or indirect reference to a buffer or select register which is to receive the data. If you use a reference, specify the prompt character c.

NOTES

a. The IBP command is similar to the IP command except that it places the input in the buffer as a single parameter and maintains embedded spaces.
b. If you do not specify a buffer reference, it uses the active input buffer.
c. The new data will always replace the parameter pointed to by the buffer pointer but does not change the position of the pointer.
d. If the user responds with RETURN only, it creates a null parameter.

EXAMPLE 1

Command	PIB Before	Input	PIB After
IBP?	AAA^BBB ^	CCC	AAA^BBB^CCC ^

EXAMPLE 2

Command	PIB Before	Input	PIB After
IBP?	AA^BB^CC ^	XX X	AA^XX X^CC ^

EXAMPLE 3

Command	PIB Before	Input	PIB After
IBP?	ABC^DEF^GHI ^	<RETURN>	ABC^^GHI ^

EXAMPLE 4

Command	File Buffer 2	Input	File Buffer 2
	Before		**After**
IBP:&2.2	000 Key	BBB	000 Key
	001 AAA		001 AAA
	002 XXX		002 BBB
	003 CCC		003 CCC

EXAMPLE 5

Command	File Buffer 2	Input	File Buffer 2
	Before	**<RETURN>**	**After**
IBP:&2.2	000 Key		000 Key
	001 AAA		001 AAA
	002 XXX		002
	003 CCC		003 CCC

JCL IF

Allows conditional execution of jCL commands based on the evaluation of an expression, or the presence (or absence) of a reference.

COMMAND SYNTAX

IF{#} reference command

or

IF reference operator expression command

SYNTAX ELEMENTS

tests for the absence of a value

reference can be a direct or indirect reference to a buffer or select register, or an A command without a surround character. Using an A command will not move a value but simply provides a reference to the parameter to be tested.

operator performs a value comparison. Operators are:

=	Equal to
#	Not equal to
<	Less than
>	Greater than
[	Less than or equal to
]	Greater than or equal to

expression follows the operator and can be one of the following:

- A direct or indirect reference to a buffer or select register.
- A string. If the string contains embedded spaces or the first character is an exclamation mark (!), percent sign (%) or ampersand (&), enclose the string in single or double quotes.
- A pattern format string. Refer to the IF (with Mask) command.

'command' is any valid jCL command.

NOTES

a. If the test result is true, it executes the command at the end of the statement. If the test yields false, it ignores command and processing continues with the next line.

b. Does not move buffer pointers if you use a direct or indirect reference to a buffer or select register.
c. Performs comparative tests on a character-by-character, left-to-right basis.

EXAMPLE

AAB is greater than AAAB and 20 is greater than 100. Use the IFN command if you want to perform numeric comparisons. You can use also perform multiple conditional tests in a single statement. For example:

```
IF %1 > A IF %1 < D T %1 is B or C
```

EXAMPLE 1

```
021 IF %1 T "%1 is not null"
022 IF # %1 T "%1 is null"
023 ...
```

Line 21 tests for a value in the first parameter of the PIB and outputs a message if the parameter is not null. Line 22 tests for opposite case.

EXAMPLE 2

```
021 IF &1.1 = ABC GO 10
022 MV %3 &1.1
023 10 ...
```

If &1.1 contains ABC execution will branch to line 23; else the value in &1.1 will be moved into %3.

EXAMPLE 3

```
010 T "Continue (Y/n) :",+
011 IP %1
012 S1
013 IF # A G 10
014 IF A(1,1) = Y G 10
015 IF A(1,1) = y G 10
016 XFinished
017 10 ...
```

If the user enters Y, y or <ENTER> to the prompt on line 11, execution will continue at label 10 on line 17. Otherwise, terminates the program

.

JCL IF E

Conditionally executes a command depending on the presence or absence of an error message after running a shell command.

COMMAND SYNTAX

IF {#} E command

IF E operator msg-key command

SYNTAX ELEMENTS

- tests for the absence of an error message
operator performs a value comparison. Operators are:

= Equal to
Not equal to

msg-key is the key of a system message from the error file.
'command' is a valid jCL command.

NOTES

a. Any system messages generated because of running a shell command (see the P command) will place the system message number in the SIB, in multivalue format. The value tested is the first multivalue (the STOP code) of the error text returned from the last command.
b. The IF E command is most often used to test for an error condition but can be used to detect any resulting system message. IF # E command tests for the absence of a system message
c. Some jCL commands, particularly those that operate on the PIB, will destroy the contents of the secondary input buffer. You should therefore use the IF E command as soon as control returns from the shell command.

EXAMPLE

```
021 HCOUNT SALES WITH VALUE > "1000"
022 PH
023 IF E = 401 G 100
```

If the SALES file does not contain any records, which match the criteria, the system generates the message

"No records counted".

Using the PH command will stop the output message at the terminal but places the message key 401 in the SIB where it can be detected by line 23.

IF (MULTIVALUED)

Compares an expression with one of several different expressions or values, and conditionally executes one of several commands.

COMMAND SYNTAX

IF reference = expression{]expression}... command{]command]}...
IF reference # expression{]expression}... command

SYNTAX ELEMENTS

reference can be a direct or indirect reference to a buffer or select register, or an A command without a surround character. Using an A command will not move a value but simply provides a reference to the parameter to be tested.
expression follows the operator and can be one of the following:

- A direct or indirect reference to a buffer or select register.
- A string. If the string contains embedded spaces or the first character is an exclamation mark (!), percent sign (%) or ampersand (&), enclose the string in single or double quotes.
- A pattern format string. Refer to the IF (with Mask) command.
- **]** represents a value mark (Ctrl])

command is any valid jCL command.

NOTES

a. The multivalues feature of the IF command enables one IF statement to be used instead of a series
b. Do not use O or X commands unless they are the last in the series. It ignores commands after an O or X.
c. The equal (=) operator will perform a logical OR on the expressions. If the reference is equal to any expression, the condition is true.
d. If more than one expression is true, it executes the command corresponding to the first true expression. If there are more expressions than commands, and the true expression does not have a corresponding command, it executes the last command in the series. If there are more commands than expressions, it ignores remaining commands.
e. If you use the not equal (#) operator, it performs a logical AND on the expressions, the reference must not equal any expression in the series for the condition to be true. In this case, it executes the first command specified and ignores subsequent commands.

If a direct or indirect reference to a buffer or select register identifies a multivalued parameter, the same jCL statement is executed regardless of which of the multivalues is true. This means that each value will not access a different command - as it would have if you had specified it directly in the IF statement.

EXAMPLE:

```
MV %1 A]B]C]D
IF %1 = X]Y]Z]C GO 10]20]30]40
```

Will cause program execution to continue from label 40

GO AND GOSUB COMMANDS

To use the special multivalued forms of the GO and GOSUB commands, you must specify one label for each result of a multiple comparison.

EXAMPLE 1

```
IF %2 = A]B]C]D GO 10]20]30]40
```

NOTE: that this is a special case of the GO and GOSUB commands. If you need to mix command types in the resulting actions, you should not use this form of the GO and GOSUB commands. You can still achieve the same effect but each result must contain a separate command.

EXAMPLE 2

```
IF %2 = A]B]C]" GO 10]GO 20]GO 30]XFinished
```

In this case, if the result of the test for null is true the program will terminate with a suitable message.

EXAMPLE 3

```
IF %1 = A]B]C G 10
If %1 is equal to A, B or C, control is transferred to label 10.
```

EXAMPLE 4

```
IF %2 # "AA"]&1.1](2N)]" GOSUB 1001
```

If %2 is not equal to AA, the content of &1.1, two numerics or null, it transfers control to subroutine 1001.

EXAMPLE 5

```
IF %3 = (#0N)](0N) GO 10]MV #4 "DOLLARS"
```

If %3 equals a pound sign (#) followed by any one or more numerics, control transfers to the statement with label 10.

If %3 equals a dollar sign ($) followed by any one or more numerics, it moves the string "DOLLARS" into the output buffer.

IF (WITH MASK)

Conditionally executes commands based on a comparison with a specified pattern.

COMMAND SYNTAX

IF reference operator (pattern) command

SYNTAX ELEMENTS

reference can be a direct or indirect reference to a buffer or select register, or an A command without a surround character. Using an A command will not move a value but simply provides a reference to the parameter for testing.
operator performs a value comparison. Operators are:

=	Equal to.
#	Not equal to.

(pattern) a format string enclosed in parentheses, which tests for a specified numeric, alphanumeric or alphabetic string. A pattern format string can consist of any combination of the following:

(nA)	Tests for n alphabetic characters. If n is 0 (zero), it assumes to match one or more alphabetic characters.
(nC)	Tests for n alphanumeric characters. If n is 0 (zero), it assumes to match one or more alphanumeric characters.
(nN)	Tests for n numeric characters. If n is 0 (zero), it assumes to match one or more alphanumeric characters.
(nP)	Tests for n printable characters. If n is 0 (zero), it assumes to match any one or more printable characters.
(nX)	Tests for n characters. If n is 0 (zero), it assumes to match any one or more characters.
(string)	Tests for one or more specified characters. If string contains numeric characters, %, #, &, !, or spaces, or any of the above patterns it must be enclosed in quotes.

EXAMPLE

The pattern (2N"-"3A"-"2N) specifies a format of two numeric characters, a hyphen, three alphabetic characters, another hyphen and then two numeric characters.

command is a valid jCL command.

NOTES

Use 0X only as the last specification in a pattern.

If you want to test for spaces in the pattern, use double quotes, for example:

IF %n = (2A" "3N" "0X)

Enclose ambiguous literal strings in quotes.

EXAMPLE 1

```
IF %3 = (3N1A) GO 10
```

If %3 matches three numerics followed by one alphabetic character, branch to label 10.

EXAMPLE 2

```
IF &1.2 # (2N"AAA") GOSUB 1001
```

If &1.2 does not equal two numerics followed by AAA, control will be transferred to subroutine 1001.

EXAMPLE 3

```
IF %1 # (2N*2A*0N) MV %1 "99*AA*1234"
```

If %1 does not equal two numerics, an asterisk, two alphabetics, an asterisk, and a number move a string that does into %1.

EXAMPLE 4

```
IF &1.1 = ("(6X)") GO 100
```

If &1.1 contains "(ABC123)", control will be transferred to label 100.

EXAMPLE 5

```
IF &1.1 = ("("0X")") GO 100
```

If &1.1 contains "any non null string", control will not be transferred to label 100.

EXAMPLE 6

```
IF &1.1 = ("("0X) GO 100
```

If &1.1 contains "a ‘(‘ followed by any non-null string”, control will be transferred to label 100.

JCL IF S

Conditionally executes a command depending on the presence or absence of an active select list.

COMMAND SYNTAX

IF {#} S command

SYNTAX ELEMENTS

- tests for the absence of an active select list

Command - is a valid jCL command.

NOTES

IF S, executes the command if there is an active select list.

IF # S executes the command if there is not an active select list

EXAMPLE

```
021 HSELECT SALES WITH VALUE > "1000"
022 PH
023 IF S G 100
```

If the SELECT command has generated an active select list, it transfers control to label 100.

JCL IFN

Allows conditional execution of commands depending on the result of numeric comparisons

COMMAND SYNTAX

IFN reference operator expression command

SYNTAX ELEMENTS

reference can be a direct or indirect reference to a buffer or select register, or an A command without a surround character. Using an A command will not move a value but simply provides a reference to the parameter to be tested.
operator performs a value comparison. Operators are:

=	equal to
#	not equal to
<	less than
>	greater than
[	less than or equal to
]	greater than or equal to

expression can be one of the following:

- A direct or indirect reference to a buffer or select register.
- A string. If the string contains embedded spaces or the first character is an exclamation mark (!), percent sign (%) or ampersand (&), you must enclose the string in single or double quotes.

command is a valid jCL command.

NOTES

a. IFN tests numbers by value, rather than their ASCII sequence of characters.
b. Ignores leading zeros, as are trailing decimal zeros. For example, if %1 contains 00123.000, IFN %1 = 123 will be true.
c. If a submitted value equates to null or is non-numeric, it uses zero. Allows leading plus or minus signs.

EXAMPLE 1

```
IFN %1 > 50 G 100
```

If %1 is greater than 50, it transfers control to label 100

EXAMPLE 2

```
IFN %1 [0 G 100
```

If %1 is less than or equal to 0, control will be transferred to label 100.

EXAMPLE 3

```
IFN A < %3 G 100
```

If the current input buffer parameter is less than %3, it transfers control to label 100.

JCL IH

Places a text string in the active input buffer, clears an existing parameter, or creates new null parameters.

COMMAND SYNTAX

IHtext
IHreference;input-conversion;
IHreference:output-conversion:
IH\

SYNTAX ELEMENTS

text is the text for placing in the active input buffer. Can be a literal (not enclosed in quotes), or a direct or indirect reference to a buffer or select register. The text must not contain subvalue marks.
reference is a direct or indirect reference to a buffer or select register.
input-conversion is a jQL input conversion to apply to the string before putting it in the buffer.
output-conversion is a jQL output conversion to apply to the string before putting it in the buffer.
\ (backslash). If there is no preceding space, it nulls the current parameter in the active input buffer. If there is a preceding space, it creates a new null parameter at the current buffer pointer position. It treats a backslash in any other position as part of the literal text.

NOTES

a. Replaces each group of one or more spaces in the text will be replaced with a single field mark, thereby creating new, separate parameters to replace the current single parameter.
b. Ignores leading and trailing spaces within text.
c. Use the IBH command if you want to insert text into the active input buffer as a single parameter with all blanks intact.
d. If the buffer pointer is at the beginning of an existing parameter, it replaces that parameter by the new text.
e. If the buffer pointer is in the middle of a parameter, it truncates that parameter from the current location and concatenates the new parameter (without a leading field mark).
f. If the buffer pointer is at the end of the input buffer, it creates one or more new parameters.
g. Does not change the position of the buffer pointer.

CREATING NULL PARAMETERS

If the buffer pointer is at the start of a parameter, IH\ will null the parameter it deletes the characters between the field marks but retains the field marks.

If the buffer pointer is in the middle of a parameter IH\ removes the remaining characters from that point to the end of the parameter; if the buffer pointer is at the end of the buffer, IH\ creates a new null parameter.
IH \ creates a new null parameter at the position pointed to by the input buffer pointer. Note the space between the H and the backslash character.

EXAMPLE 1

Command	PIB Before	PIB After
IHXXX	AB^CD^YY^ZZ ^	AB^CD^XXX^ZZ ^

EXAMPLE 2

Command	PIB Before	PIB After
IH GH IJ	AB^CD^EF ^	AB^CD^EF^GH^IJ ^

EXAMPLE 3

%3 contains

```
9873
```

Command	PIB Before	PIB After
IH%3:D2:	AB^CD^9873^GH ^	AB^11^JAN^95^9873^GH ^

EXAMPLE 4

Command	PIB Before	PIB After
IH%4	AB^CD^EF^GH IJ^ ^	AB^CD^EF^GH^IJ^ ^

This example demonstrates how, in effect, you can replace a space with a field mark within a parameter. It copies the fourth parameter of the PIB back into the same location but replaces the space with a field mark.

EXAMPLE 5

Command	PIB Before	PIB After
IH\	AB^CD^EF^GH ^	AB^CD^^GH

EXAMPLE 6

Command	PIB Before	PIB After
S(7)		
IH\	AB^CDEFGH^IJK ^	AB^CDE^IJK

This example demonstrates how to truncate a parameter.

EXAMPLE 7

Command	PIB Before	PIB After
IH \	AB^CD^EF^GH ^	AB^CD^^EF^GH ^

EXAMPLE 8

Command	PIB Before	PIB After
IH \	AB^CDEFGH^IJK ^	AB^CDE^^FGH^IJK

JCL IN

Prompts for input and places it in the secondary input buffer. Selects the secondary input buffer as the active buffer input

COMMAND SYNTAX

IN{c}

SYNTAX ELEMENTS

c is an optional prompt character, which, once used, remains in effect until the issue of a new IBN, IBP, IN or IP command. If c is not specified, the prompt character will default to the last used prompt character, or to a colon (:).

NOTES

a. The new data replaces the content of the SIB, which will remain active until an RI, S(n) or MV %n source command is used.
b. Removes leading and trailing spaces and groups of one or more embedded spaces are replaced by a single field mark. Use the IBN command if you want to maintain embedded spaces.
c. If the user responds with ENTER only, it creates a null parameter.
d. Upon completion of the command, it positions the buffer pointer at the beginning of the buffer.

EXAMPLE 1

Input	SIB Before	SIB After
ABC	^	ABC ^

EXAMPLE 2

Input	SIB Before	SIB After
ABC DEF	XYZ ^	ABC^DEF

EXAMPLE 3

Input	SIB Before	SIB After
<ENTER>	WWW^XXX ^	

JCL IP

Prompts for input and places it into the active input buffer or a nominated buffer.

COMMAND SYNTAX

IP{c{r}}

SYNTAX ELEMENTS

c is an optional prompt character, which, once used, remains in effect until a new IBN, IBP, IN or IP command is issued. If c is not specified, the prompt character will default to the last prompt character used, or to a colon (:).

r is a direct or indirect reference to a buffer or select register which is to receive the data. If you use a reference, you must specify the prompt character c.

NOTES

a. If you do not specify a buffer reference, it uses the active input buffer.
b. The new data will replace the parameter at the current buffer pointer position but will not move the pointer.
c. Removes leading and trailing spaces and replaces groups of one or more embedded spaces by a single field mark. By replacing a parameter with data that contains spaces, you can insert several new parameters.
d. When you place data containing embedded spaces into a file buffer, the new parameters will replace successive fields in the buffer. For example, if the response to an IP?&2.1 command is: <SPACE>AA<SPACE><SPACE>BB<SPACE>CC" fields one, two, and three, of file buffer 2 will be replaced with "AA", "BB", and "CC".
e. If the user responds with RETURN only, it creates a null parameter.
f. If you want to keep the input data exactly as entered, use the IBP command.

EXAMPLE 1

Command	PIB Before	Input	PIB After
IP?	AAA^BBB ^	CCC	AAA^BBB^CCC ^

EXAMPLE 2

Command	PIB Before	Input	PIB After
IP?	AA^BB^CC ^	XX X	AA^XX^X^CC ^

EXAMPLE 3

Command	PIB Before	Input	PIB After
IP?	ABC^DEF^GHI ^	<ENTER>	ABC^^GHI ^

EXAMPLE 4

Command	File Buffer 2 Before	Input	File Buffer 2 After
IP:&2.2	000 Key 001 AAA 002 XXX 003 CCC	BBB	000 Key 001 AAA 002 BBB 003 CCC

EXAMPLE 5

Command	File Buffer 2 Before	Input	File Buffer 2 After
IP:&2.2	000 Key 001 AAA 002 XXX 003 DDD	BB CC DD	000 Key 001 AAA 002 BB 003 CC 004 DD

JCL IT

Reads a tape record into the primary input buffer.

COMMAND SYNTAX

NOTES

a. The IT command will read a tape record into the primary input buffer.

b. It places the new data at the beginning of the buffer and replaces all existing buffer data.

EXAMPLE

Command	PIB Before	PIB After
IT	ABC	tape data

JCL L

Formats printed output.

COMMAND SYNTAX

L element{, element}...

SYNTAX ELEMENTS

element can be any of the following:

"text"	Prints literal text. Enclose the text in quotes.
r{;input;}	Prints the result of a direct or indirect reference to a buffer or select register (r). If required, apply a jQL input conversion before output.
r{:output:}	Prints the result of a direct or indirect reference to a buffer or select register (r). If required, apply a jQL output conversion before output.
n	Skips n lines before printing. You cannot use n in a HDR print format specification.
(c)	Sets the print position to column number c. Can be a direct or indirect reference to a buffer or select register.
+	Suppresses NEWLINE being output at the end of the L command. You cannot use + in a HDR print format specification.
C	Closes the print file and forces printing. You cannot use C in a HDR print format specification.
E	Ejects to top-of-form (form feed). You cannot use E in a HDR print format specification.
N	Redirects subsequent printer output to the terminal. Normally only used for debugging. Must be the only element in an L command.
HDR	Allow you to define a page heading. HDR, must be the first element in the L command.
L	Outputs a line feed in the heading. Only used in a HDR specification.
P	Outputs the current page number in a heading. Only used in a HDR specification.
T	Outputs current time and date in a heading. Only used in a HDR specification.
Z	Resets the current page number in a heading to zero. Only used in a HDR specification.

NOTES

a. Output from the L command creates (or adds to) a print file.

b. The print file will remain open until the issue of a shell command by using the P command.
c. If the shell command also generates print output, it adds it to the same print file.
d. You can close the print file and avoid any unnecessary output by choosing any shell command, which does not generate print output, or by using the L C command. Alternatively, you can hold the print file open indefinitely by using the O option of the SP-ASSIGN command.
e. Create continuation lines (which do not start with an L) by terminating each preceding line with a comma (,).
f. A + specified as the last element the command will cause the output from the next L command to be appended.
g. If you specify a heading statement (HDR), which contains direct or indirect references, it evaluates these (and included in the heading) as it processes the command. Subsequent changes to the source values will have no effect on the heading.

EXAMPLE

```
MV %1 "Quarter 4"
L HDR,"PAGE ",P,(10),T
L 5,"Sales Report for ",%1
```

Output:

```
PAGE 1 10:25:17 10 OCT 1999

Sales Report for Quarter 4
```

JCL M

Marks a destination for a GO F or GO B command

COMMAND SYNTAX

M

NOTES

a. Use the M command with the GO B and GO F branch commands to mark the destination of the jump.
b. During execution of a jCL program, it "remembers" the location of the last M command it passes through. When encountering a GO B command, the program can then branch straight back to the remembered location. Remember you must execute an M command for it to be remembered.
c. GO F scans forward from the current location until it finds an M command and then resumes execution from there.
d. If grouping commands on a line, the M command must be the first command on the line.

EXAMPLES

See the GO B and GO F commands for examples of usage

JCL MS

Move the entire content of the secondary input buffer to the primary input buffer.

COMMAND SYNTAX

MS

NOTES

a. MS copies the entire content of the secondary input buffer and inserts the parameters before the parameter currently pointed to in the primary input buffer.
b. The primary input buffer must be active when executing MS. After the move, the secondary input buffer will be empty.
c. Hold file numbers are returned as Entry #n, where "n" is the hold file number.

EXAMPLE

```
001 PQN
002 HSP-ASSIGN HS
003 P
004 HCOPY SALES ABC (P
005 P
006 S10
008 MS
...
```

The COPY command on line 4 creates a print file and writes the hold file number to the SIB. The S10 command on line 6 positions the PIB buffer pointer to parameter 10. The MS command on line 8 moves the contents of the SIB into the PIB starting at the 10th parameter of the PIB.

JCL MV

Copies data between buffers or between buffers and select registers

COMMAND SYNTAX

MV destination source{,source}...{,*{n}}{,_}

or

MV destination source{*source}...

SYNTAX ELEMENTS

destination	is a direct or indirect reference to the destination buffer or select registers which is to receive that data.
source	is the data you want to copy. Can be a direct or indirect reference to a buffer or select register, or a literal string enclosed in single or double quotes.
,*	copies all source parameters starting with the specified parameter. If ***** is the last operand in the source field, the destination buffer or select register will be truncated after the last copied parameter.
,*n	copies the number of source parameters specified by **n** starting with the specified parameter. The destination buffer or select register will not be truncated.
,_	specifies that the destination for truncating after copying the source.
***source**	specifies the source values for concatenating into one field in the destination buffer or select register.

NOTES

a. If the source is a literal string containing just two double quotes, it nulls the destination.
b. If the input buffer (%n) is specified as the destination, it will be selected as the active input buffer and the buffer pointer will be left at the beginning of any copied data.
c. If the field or parameter number in destination is larger than the current number of fields or parameters, it creates intervening null values automatically.
d. Specify an asterisk (*) as the last character in the source element if you want to copy all the following source buffer parameters.

EXAMPLE 1

```
MV &2.2 %1,*
```

Copies the first parameter of the PIB to field 2 of file buffer 2 Copies parameter 2 of the PIB to field 3 etc

If you want to copy a series of parameters, specify the number of additional parameters for copying after the asterisk.

EXAMPLE 2

```
MV &2.2 %1,*3
```

Will copy the first, second, third and fourth parameters of the PIB into fields 2, 3, 4 and 5 of file buffer 2; will not change the remainder of file buffer 2.
If you specify a series of values as the source, each value will copied to successive locations in the destination.

EXAMPLE 3

```
MV %2 "ABC",&2.1,!1
```

Will concatenate ABC, field 1 of file buffer 2 and the next value from select register 1, and place the result into PIB parameter 2.
Using commas as placeholders in the source can preserve intervening parameters in the destination.

EXAMPLE 4

```
MV %2 "ABC"*&2.1*!1
```

Will copy ABC into PIB parameter 2, field 1 of file buffer 2 into PIB parameter 3, and the next value from select register 1 into PIB parameter 4
Will concatenate two or more source values separated with an asterisk (*) into a single parameter in the destination

EXAMPLE 5

```
MV %1 "ABC",,,"DEF"
```

Would copy ABC into PIB parameter 1 and DEF into PIB parameter 4; PIB parameters 2 and 3 would not be affected.

EXAMPLE 1

Command	PIB Before	PIB After
`MV %5 "XXX"`	ABC	ABC^^^^XXX

EXAMPLE 2

Command	PIB Before	PIB After
MV %4 %1	ABC^DEF^GHI	ABC^DEF^GHI^ABC

EXAMPLE 3

PIB contains:

```
QTR^ABC^DEF
```

Command	POB Before	POB After
MV #3 %1	SORT^SALES^	SORT^SALES^QTR^

EXAMPLE 4

Command	PIB Before	PIB After
MV %1 "AA",,"CC"	XX^BB^YY^ZZ	AA^BB^CC^ZZ

EXAMPLE 5

File buffer 2 contains:

```
000 Key

001 111
002 AAA
003 BBB
```

Command	File Buffer 1 Before	File Buffer 1 After
MV &1.1 &2.2,*	000 KEY1	000 KEY1
	001 WWW	001 AAA
	002 XXX	002 BBB
	003 YYY	003 YYY
	004 ZZZ	004 ZZZ

EXAMPLE 6

PIB contains:

```
ABC^DEF^GHI^JKL^MNO
```

Command	File Buffer 1 Before	File Buffer 1 After
MV &2.1 %3,_	000 Key 001 XXX 002 YYY	000 Key 001 GHI

JCL MVA

Copies a value from the source to the destination buffer and stores it as a multivalue.

COMMAND SYNTAX

MVA destination source

SYNTAX ELEMENTS

destination is a direct or indirect reference to a buffer or select register which is to receive the data.

source is the data for copying. The source can be a direct or indirect reference to a buffer or select register, or a literal string.

NOTES

a. Copies new values to the destination in ascending ASCII sequence.
b. Will not copied a new value if it already exists in the destination buffer.
c. Copies the source data if multivalued, to the destination without modification. This might create duplicate values and invalidate the ascending sequence.
d. If the destination is the input buffer, it will leave the buffer pointer at the beginning of the destination parameter.

EXAMPLE 1

PIB contains:

```
ABC^DEF^GHI
```

Command	File Buffer 1 Before	File Buffer 1 After
MVA &1.1 %3	000 Key 001 FFF]HHH 002 YYY	000 Key 001 FFF]GHI]HHH 002 YYY

EXAMPLE 2

File buffer 2 contains:

```
000 Key
001 GG]YY
002 AAA
```

Command	File Buffer 1 Before	File Buffer 1 After
MVA &1.1	000 Key	000 Key
&2.1	001 FFF]HHH	001 FFF]GG]YY]HHH
	002 YYY	002 YYY

JCL MVD

Deletes a value from a multivalued parameter in the target buffer

COMMAND SYNTAX

MVD target source

SYNTAX ELEMENTS

target is a direct or indirect reference to a buffer or select register which contains the data for deletion.

source is the data you want to delete. Can be a literal string or a direct or indirect reference to a buffer or select register.

NOTES

a. Values in the target must be in ascending ASCII sequence otherwise the result of the command will be unpredictable.

b. If the source does not exist or cannot match the multivalue, the command has no effect, except perhaps to move the buffer pointer.

c. If the source element exists more than once in the target parameter, it deletes only the first occurrence.

d. If the target is the primary input buffer, it leaves the buffer pointer at the beginning of the specified target parameter.

EXAMPLE

Command	File Buffer 1 Before	File Buffer 1 After

`MVD &1.1 DEF`	`000 Key`
	`001 AAA]DEF]GHI`
	`002 YYY`
		`000 Key`
		`001 ABC]GHI`
		`002 YYY`

JCL O

Outputs a text string to the terminal

COMMAND SYNTAX

O{text}{+}

SYNTAX ELEMENTS

text is the text for display.

+ inhibits a NEWLINE at the end of the output and leaves the cursor positioned to the right of the last character displayed; often used when prompting for input.

NOTES

a. The O command has Largely replaced by the T command, the O command also provides cursor positioning and special display features.
b. A blank line will be output where there is no supplied text.

EXAMPLE 1

Command	Terminal output
OSALES SYSTEM	SALES SYSTEM

EXAMPLE 2

Command	Terminal output
O Enter Password + IP:	Enter Password:

JCL P

Submits the shell command created in the primary output buffer for execution

COMMAND SYNTAX

P{P}{H}{Ln}{X}{U}{W}

SYNTAX ELEMENTS

P displays the primary and secondary output buffers. In ROS emulation mode, displays the command and prompts to continue. Normally only used for testing or debugging

H suppresses (hushes) any terminal output normally displayed; combine the H and Ln options as PHLn.

Ln sets an execution lock where n represents a lock number from 0 to 255. The lock is after execution of the command. It forces any other process attempting to set the same lock to wait; combine the H and Ln options as PHLn.

X terminates the program after executing the command; it cannot be used with any other options.

U specifies that UNIX submit the command for execution.

W causes the command to behave like PP when used in a non-ROS emulation.

NOTES

a. When executing the P command, it passes control to the shell and only returns upon completion of the shell process.
b. After executing the P command, it clears both output buffers and turns off the stack.
c. Makes commands and data in the secondary output buffer available to processes, which require further input.
d. If you need to preserve buffer contents when passing control between jCL programs, use ( ) or [] command instead.
e. If you use the PP variants (PP, PPH, and PPLn), it will display the content of both output buffers before executing; it will prompt you with a question mark (?). Enter:

Y - to continue

S - to cancel execution but continue the program.

N - to cancel execution and exit the program

EXAMPLE 1

```
003 HLIST SALES QTR VALUE
004 P
```

Copy the jQL command LIST SALES QTR VALUE to the output buffer and execute it.

EXAMPLE 2

```
003 HCOPY SALES ABC
004 STON
005 H(SALES.HOLD<
006 PP
```

Place the COPY command in the primary output buffer. Turn the stack on. Put the response to the TO: prompt into the secondary input buffer. Issue the PP command to display the contents of the buffers and prompt for input before continuing.

EXAMPLE 3

```
003 Henv | grep EMULATE
...
006 PU
```

Issue the UNIX grep command.

JCL RI

Resets (clears) the primary and secondary input buffers.

COMMAND SYNTAX

RI

RIp

RI(n)

SYNTAX ELEMENTS

p specifies starting parameter from which to clear to the end of the buffer. Can be a number or a direct or indirect reference to a buffer or select register.
(n) specifies the starting column from which to clear to the end of the buffer; can be a number or a direct or indirect reference to a buffer or select register.

NOTES

a. The RI command clears the entire PIB and SIB.

- RIp clears the PIB starting from parameter p and continuing to the end of the buffer.
- RI(n) clears the PIB starting from parameter n and continuing to the end of the buffer.

b. It leaves the buffer pointer at the end of the PIB. The primary input buffer becomes the active buffer and clears the secondary input buffer.

EXAMPLE 1

Command	PIB Before	PIB After
RI	ABC^DEF^GHI ^	^

EXAMPLE 2

Command	PIB Before	PIB After
RI3	ABC^DEF^GHI	ABC^DEF ^

EXAMPLE 3

Command	PIB Before	PIB After
RI(6)	ABC^DEF^GHI	ABC^D ^

JCL RO

Resets (clears) the active output buffer

COMMAND SYNTAX

RO

NOTES

a. The RO command clears the active output buffer.

b. Leaves the buffer pointer at the beginning of the buffer.

EXAMPLE 1

Command	POB Before	POB After
STOFF		
RO	ABC^DEF	

EXAMPLE 2

Command	SOB Before	SOB After
STON		
RO	GHI^JKL	

JCL RSUB

Terminates execution of a local subroutine and returns control to a statement line following the GOSUBcommand that called the subroutine

COMMAND SYNTAX

RSUB {n}

SYNTAX ELEMENTS

n specifies return of control to the n'th statement line after the calling GOSUB. Can be a number or a direct or indirect reference to a buffer or select register.

NOTES

a. If no n is specified, control will return to the statement immediately following the calling GOSUB.
b. It ignores an RSUB without a corresponding GOSUB.

EXAMPLE 1

```
010 GOSUB 1001
011 ...
012 ...
.
051 1001 T "Press <CR> to continue...",+
052 S10
052 IP?
053 RSUB
```

The RSUB command on line 53 will return control to line 11.

EXAMPLE 2

```
010 GOSUB 1001
011 ...
012 ...
.
051 1001 T "Press <CR> to continue...",+
052 S10
052 IP?
053 RSUB 2
```

The RSUB command on line 53 will return control to line 12.

JCL RTN

Terminates execution of an external subroutine and returns control to a statement following the [ ] command that called the subroutine

COMMAND SYNTAX

RTN{n}

SYNTAX ELEMENTS

n specifies return of control to the n'th statement line after the calling [] command. Can be a number or a direct or indirect reference to a buffer or select register.

NOTES

a. If no n is specified, control will return to the statement immediately following the calling [] command.
b. A RTN without a corresponding [] command terminates the program.

EXAMPLE 1

MENU	MENU2
051 [SUBS MENU2]	066 RTN
052	
053	

jCL program MENU calls MENU2 from line 51. When MENU2 reaches line 66, it returns control to MENU at line 52.

EXAMPLE 2

MENU	MENU2
051 [SUBS MENU2]	066 RTN 2
052	
053	

JCL program MENU calls MENU2 from line 51. When MENU2 reaches line 66, it returns control to MENU at line 53.

JCL S

Positions the primary input buffer pointer to a specified parameter or column, and selects the primary input buffer as active.

COMMAND SYNTAX

Sp
S(n)

SYNTAX ELEMENTS

p specifies the parameter to which the buffer pointer should be set. Can be a number or a direct or indirect reference to a buffer or select register.
(n) specifies the starting column to which the buffer pointer should be set. Can be a number or a direct or indirect reference to a buffer or select register.

NOTES

a. Sp sets the buffer pointer to the field mark preceding parameter p.
b. If the specified column or parameter number is less than 2, it places the buffer pointer at the beginning of the active input buffer.
c. If the specified column or parameter number is beyond the end of the buffer, it positions the buffer pointer at the end of the buffer.
d. Use the MV command to create parameters beyond the current end of the buffer.
e. Clears all data in the secondary input buffer.

EXAMPLE 1

Command	PIB Before	PIB After
S3	ABC^DEF^GHI ^	ABC^DEF^GHI ^

EXAMPLE 2

Command	PIB Before	PIB After
S(4)	12345^ABC	12345^ABC ^

JCL STOFF

Selects the primary output buffer as the active output buffer

COMMAND SYNTAX

STOFF

ST OFF

NOTES

a. The stack can be turned on (STON) or off (STOFF) at any point within a jCL program.

b. At the start of a jCL program or after execution of an RO or P command, both output buffers will be empty, and the stack will be off.

c. When the stack is off, H commands will place their data in the primary output buffer.

JCL STON

Selects the secondary output buffer as the active output buffer

COMMAND SYNTAX

STON

ST ON

NOTES

a. The STON command selects the SOB as the active output buffer. With the stack turned on, it places all data moved to an output buffer with an MV, H or A command in the secondary output buffer.

b. Use the stack to feed responses to interactive processes. Use also to store a series of commands that you might need for example, when you are dealing with select lists. For example, do a GET-LIST, followed by a SORT-LIST and then run a jBASIC program.

c. Typically, you would create the command necessary to start the external job stream in the primary output buffer. Next, you would turn the stack on and store all the necessary responses (or commands) to the external process. When you issue the P command to execute the external program, instead of taking its input from the terminal, it feeds the program directly from the stack.

d. Terminated successive responses in the stack, with a less-than-character (<) to represent a RETURN key depression. A single response or the last response in the stack does not require the less than character (<) because it generates a RETURN by the P command.

EXAMPLE

Command	POB Before	POB After
STON H (SALES.HOLD	COPY^SALES^ABC SOB Before	COPY^SALES^ABC SOB After (SALES.HOLD<

JCL T

Produces formatted terminal output.

COMMAND SYNTAX

T element{, element}

SYNTAX ELEMENTS

element is literal text, a reference or a formatting instruction:

-

"text"	Outputs the specified text; enclose the text in single open quotes.
r{;input;}	Outputs the value obtained by the direct or indirect reference to a buffer or select register specified by r. Apply an optional jQL input conversion to the value before output.
r{:output:}	Outputs the value obtained by the direct or indirect reference to a buffer or select register specified by r. Apply an optional jQL output conversion to the value before output.
(c,r)	Sets the cursor to the column c and row r. Can be direct or indirect buffer references.
(c)	Sets the cursor to the column c in the current row.
***cn**	Outputs the character c n number of times. Value n can be a direct or indirect reference to a buffer or select register.
(-n)	Provides terminal independent cursor control or video effects.
+	Output carriage return/line feed at the end of the output line. On ROS emulations, this clause will inhibit the carriage return/line feed at the end of the output line.
B	Sounds terminal bell.
C	Clears the screen (outputs top-of-form).
D	One-second delay. Often used when outputting error messages.
Ir	Converts the integer r (0 to 255) into its equivalent ASCII character. r can be a direct or indirect reference to a buffer or select register that contains the integer.
L	Terminates a loop started with a T element. Executes the elements between the T and L three times.
Sn	Outputs the number of spaces specified by n. The value n can be a direct or indirect reference to a buffer or select register than contains the number of spaces.
T	Marks the top of a loop. Terminates the loop by the L element. Executes the elements between T and L three times.

U	Moves the cursor up one line.
Xr	Converts the hex value r (x"00" to x"FF") into its equivalent ASCII character. The value r can be a direct or indirect reference to a buffer or select register that contains the hex value.

NOTES

Follow the T command by a single space and a comma, which must separate each element. Create continuation lines (which do not start with a T) by ending the preceding line with a comma.

TERMINAL INDEPENDENT CURSOR CONTROL

Terminal independent cursor control is available using the same table of negative numbers as used by the jBASIC @ command. See the jBASIC @ command for more details.

EXAMPLE 1

Commands	Terminal Output
`MV %1 "99"` `T C, "%1 = ",%1`	`clear screen` `%1 = 99`

EXAMPLE 2

Commands	Terminal Output
`T "Enter Password :",+` `IP %3`	`Enter Password : _`

EXAMPLE 3

Command	Terminal Output
`T *=6`	`======`

EXAMPLE 4

Commands	Terminal Output
`MV %1 "9873"` `T "DATE:",S2,%1:D2:`	`DATE: 11 JAN 95`

EXAMPLE 5

Command	Terminal Output
`T(0,10),T,(0),"ERROR",B,D,(O),S5,D,L`	`ERROR bell (flashing)`

EXAMPLE 6

Commands	Terminal Output
T X1B,"J",+	erase to end of screen
T I27,I74,+	erase to end of screen
T (-3),+	erase to end of screen

JCL TR

Traces jCL program execution and displays each command (source line) before executing.

COMMAND SYNTAX

TR {ON}

TR OFF

NOTES

a. The TR command will help you to test and debug your jCL programs.
b. TR or TR ON turns the trace on. TR OFF turns the trace off.
c. You can easily provide a general-purpose trace program by creating a jCL program called TRACE in the file defined by the JEDIFILENAME_MD environment variable. The TRACE program should look like this:

```
TRACE

001 PQN
002 TR ON
003 (%1 %2)
```

Run the TRACE program like this:

TRACE jcl_file jcl_program

If the target program relies on the initial content of the PIB, you will have to insert a line into the program like this:

```
002 MV %1 %2,*
```

To move the PIB parameters to their required positions - otherwise the word TRACE will appear in %1 and the jCL program name (normally in %1) will be in %2.

EXAMPLE

```
001 PQN
002 TR
003 MV %21 NULL ," "
004 MV %98 1,A
005 S21
006 U147F
007 T MTS
008 S23
009 U147F
010 D D2
011 T "Today"s date is ", %23
012 T "the time is ",%21
013 TR OFF
```

Will output:

```
MV %21 ","," "
MV %98 1,A
S21
U147F
S23
U147F
T "Today"s date is ", %23
Today"s date is 01/01/95
T "the time is ",%21
```

The time is 19:30:32

Line 2 turns the trace on. Line 13 turns it off.

JCL U

Executes a "user exit" from a jCL program

COMMAND SYNTAX

Unxxx

SYNTAX ELEMENTS

n represents the user exit entry point

xxx is the id of the user exit.

NOTES

a. User exits are user written functions, used to manipulate buffers and perform tasks beyond the scope of the standard jCL commands.

b. See the Time and Date topic and the TR command for more examples of "standard" user exits.

EXAMPLE 1

```
012 U31AD
Returns the current port number
```

EXAMPLE 2

```
012 U307A
013 300
Causes the process to "sleep" for 300 seconds
```

EXAMPLE 3

```
003 U407A
004 12:10
Causes the process to "sleep" until 12:10
```

JCL X

Halts execution of the program and returns control to the shell

COMMAND SYNTAX

X{text}{+}

SYNTAX ELEMENTS

text is any text for display on exit.

+ suppresses a NEWLINE at exit or after text output.

NOTES

The X command returns control directly to the shell.

EXAMPLE 1

```
F-OPEN 1 SALES
XCannot Open SALES file!
```

The X command stops execution of the program if the file SALES cannot be opened, and displays a suitable message.

LIST PROCESSING COMMANDS

The two shell commands associated with jCL which permit list handlings are the PQ-SELECT and PQ-RESELECT commands.

See the Manual ‘List Processing’ (Programmers Reference Manual) for more information on lists.

PQ-RESELECT

Executed from a jCL program, resets the pointer of a specified select register to the beginning of the list of record keys.

COMMAND SYNTAX

PQ-RESELECT register-number

SYNTAX ELEMENTS

register-number is the number (1 to 5) of the select register for resetting.

NOTES

a. Execution of this command is from the primary output buffer to reset the pointer of a specified select register back to the beginning of the list.
b. Each time you use the "!" reference to retrieve a value from the list the value is not destroyed. Simply advances the pointer to the next parameter in the list. PQ-RESELECT resets the pointer back to the beginning of the list so that you can perform another pass.
c. You cannot execute PQ-RESELECT directly from the shell.

EXAMPLE

```
HSELECT SALES
STON
HPQ-SELECT 1
PH
MV %1 !1
IF # %1 XNo records selected
HPQ-RESELECT 1
PH
10 MV %1 !1
IF # %1 XFinished
...
GO 10
```

PQ-SELECT

Executed from a jCL program, loads a list of keys into a select register

COMMAND SYNTAX

PQ-SELECT register-number

SYNTAX ELEMENTS

register-number is the number of the select register (1 to 5) in which to place the keys.

NOTES

a. To use PQ-SELECT you must first construct a list by using one of the list processing commands such as SELECT, SSELECT, QSELECT, BSELECT, GET-LIST, FORM-LIST, SEARCH or ESEARCH. Place the PQ-SELECT command in the stack for processing as part of the external job stream when the required list is active.
b. Retrieve the list elements one at a time, using a "!n" direct or an indirect reference.
c. You cannot execute PQ-SELECT directly from the shell.

EXAMPLE

```
001 PQN
002 HSSELECT SALES
003 STON
004 HPQ-SELECT 1
005 P
006 10 MV %1 !1
007 IF # %1 X Done
008 T %1
009 GO 10
```

This example selects all the records in the SALES file, loads the record-list into select register 1 and displays the keys one at a time.

CHAPTER 5 PARAGRAPHS

You can only invoke paragraphs automatically via the jshell, jsh, when executing in jsh mode. The jshell, in jsh mode, will attempt to read the first parameter of an entered command from the MD file as

previously specified by the JEDIFILENAME_MD environment variable. If the record is determined to be a paragraph record, it invokes the paragraph interpreter, para. You can also invoke a logon paragraph record by the jshell via the dash option; alternatively, invoke the paragraph record from the command line using the paragraph interpreter.
e.g. para ./ PARAGRAPHX

Format

The first line of a paragraph record begins with PA e.g.

```
PARAGRAPHX
001 PA
```

Follow this line by any of the following:

Comment

Any Line preceded by an asterisk. e.g:

* THIS IS A COMMENT LINE

Label

Any non-spaced text suffixed by a colon. e.g

Label:

Paragraph Prompt

A paragraph prompt to obtain values, e.g.:
<<I2,Second,1N>>

Load parameter two from the command line into PromptId ‘Second’. If parameter two on the command line is null then prompt for a value for ‘Second’ ensuring it validates to one numeric.
Format the paragraph prompt as follows:
<<{Code,}PromptId{,Mask}>>

Code

A	Always Prompt.
Cn	Use parameter n from command line. If value does not exist then null is used.
In	Use parameter n from command line. If value does not exist then prompt.

Sn	Use parameter n from original command line. If value does not exist then prompt.
P	Use input for all PromptIds of the same name. This is the default action.
R{s}Prompt	use s as separator, until null input. Default separator is space.
F (File, RecordId {,Attr {,Value {,Subvalue}}})	Input from record RecordId, in file File.
@(c,r)	Prompt at column c, row r.
@(BELL)	Sound BELL at prompt.
@(CLR)	Clear screen before prompt.
@(TOF)	Prompt at top left of screen.

PromptId	Identifier used to name prompt values.
Mask	Validate Input

Pattern Match

e.g. 0N - Match any numerics

- 7X - Match seven printable characters.
- 3A - Match three alpha characters.
- text - Match text

(Conversion)
e.g. (D2/) - Match date dd/mm/yy or mm/dd/yy

~~Mask means input should NOT match mask.
Note: A Paragraph prompt not mixed with a command line or paragraph statement will attempt to execute the resultant value.

```
001 PA
002 <Command>
```

Entering: List filename will execute the specified command. E.g List Filename

Paragraph Statement

One of the following syntax statements

DISPLAY text	Output text
DATA Input	Stacked input
GO Label	Continue at Label
LOOP	Start Loop
REPEAT	Repeat Loop
CLEARPROMPTS	Clear Prompt Identifier Values
IF Expression	THEN Statement

Where:

Expression - Includes PromptId, operators and/or @variables

@DATE	Current internal date	
@TIME	Current internal time	
@DAY	Current two digit day of month	
@MONTH	Current two digit month of year	
@YEAR	Current four digit year	
@LOGNAME	Environment PLID value	SYSTEM (49)
@USERNO	Current port number	SYSTEM (18)
@WHO	Current user name	SYSTEM (19)
@ABORT.CODE	Last error code	SYSTEM (17)
@SYSTEM.RETURN.CODE	Last error code	SYSTEM (17)
@USER.RETURN.CODE	TBD	

Operators include MATCHES, EQ, LE, LT, etc.

COMMAND

Any executable command with or without paragraph prompts. E.g:

```
SSELECT<<I2,FILEX,10X>>
or
SELECT FILEX
```

EXAMPLES

```
PARAGRAPHX
001 PA
002 IF <<C3,THIRD>> EQ “” THEN GO CHK.SECOND
003 DISPLAY GOT 3 ARGUMENTS
004 GO END
004 CHK.SECOND:
```

```
005 IF <<C2,SECOND>> EQ “” THEN GO END
006 DISPLAY GOT 2 ARGUMENTS
007 END:

PARAGRAPHX
001 PA
002 * This is an example paragraph
003 IF<<C2,COMMAND>> NE “” THEN GO DO.CMD
004 DISPLAY ENTER
005 CLEARPROMPTS
006 DO.CMD:
007 DISPLAY ENTER

001 PR
002 filename
003 paragraph (default to JEDIFILENAME_MD)
```

INDEX

E

F

G

H

I

J

L

M

O

P

Q

R

S

T

U

X