TEMENOS™

jBASE Query Language

Programmers Reference Manuals

jBASE 4.1

Copyright

Acknowledgements

Information regarding Unicode has been provided in part courtesy of the Unicode Consortium. The Unicode Consortium is a non-profit organization founded to develop, extend and promote use of the Unicode Standard, which specifies the representation of text in modern software products and standards. The membership of the consortium represents a broad spectrum of corporations and organizations in the computer and information processing industry. The consortium is supported financially solely through membership dues. Membership in the Unicode Consortium is open to organizations and individuals anywhere in the world who support the Unicode Standard and wish to assist in its extension and implementation.
Portions of the information included herein regarding IBM's ICU has been reprinted by permission from International Business Machines Corporation copyright 2001

Errata and Comments

If you have any comments regarding this manual or wish to report any errors in the documentation, please document them and send them to the address below:

Technical Publications Department
TEMENOS UK Limited
6th Floor Kodak House
Hemel Hempstead
Hertfordshire
HP1 1JY
England
Tel SB: +44 (0) 1442 411800
Direct +44 (0) 1442 411808
Fax: +44 (0) 1442 411900

Please include your name, company, address, and telephone and fax numbers, and email address if applicable. documentation@temenos.com

Contents

Organization of This Manual

This manual contains the following:

Chapter 1	provides an overview of jQL, describes the sample database, and demonstrates how to enter simple queries.
Chapter 2	covers how to structure a query, select records, sort the output results, and look at the internal form of your data.
Chapter 3	explains how you can customize your output using EVAL expressions, aggregate functions, breakpoints, field qualifiers, and report qualifiers.
Chapter 4	discusses how to create and use select lists.
Chapter 5	shows how to redirect output to files and tape instead of to the terminal screen or printer.
Appendix A	contains the file dictionaries for the 10 files making up the sample database used in the examples in this document.

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 Introduction

The jBASE Query Language (jQL) is a powerful and easy to use facility, which allows you to retrieve data from the database in a structured order and to present the data in a flexible and easily understood format. You can enter jQL Commands from your terminal or embed jQL Commands in applications programs, procs and paragraphs to access data in jBASE files. The language is characterized by the use of intuitive Commands that resemble everyday English language Commands.
You might for instance manage a retail department and need to review a particular set of figures, which requires the phrase: “Show me the sales figures for January sorted in date order.”
The jQL Command would look like this:

```
LIST SALES WITH MONTH = "JANUARY" BY DATE
```

By using the jQL Command LIST with a file named SALES and your predefined data definition records such as MONTH and DATE, you can construct complex ad-hoc reports directly from the Command line interface (>). You can also choose how you want the information presented; displayed directly to your printer or to your screen; listed in date order, or in descending or ascending order. The choice is yours as jQL contains a rich range of commands for listing, sorting, selecting and controlling the presentation of your details and is a safe language for end users.
With the exception of the “EDELETE” Command, jQL will not alter the contents of the source data files.
All jQL Command sentences begin with a verb-like Command such as LIST or SELECT, followed by a file name such as SALES or PERSONNEL, and then a series of qualifiers and modifiers with which you control elements such as eligible data, report formatting, any totals that you want to appear and so on.
Most data files on the system will have two assigned storage areas:

1. For the data (the data section) and
2. For the data definition records (the dictionary section)

Some files might be single level and others might have multiple data sections. (See the File Management chapter of the System Administrators Guide for more details)

Data definition records kept in the dictionary portion of the file defines all the data fields in a file. These data definition records do not have to exist (you can use defaults provided in the environment variables or even the dictionaries of other files). However, where you need to manipulate ‘say’ dates

(which are held in internal format), or to join data held in different files, you will find that one or more definition records will be required for each data field.
The data definition records are simple to create and maintain.
FOR EXAMPLE:

- They allow you to specify the position of the data in a record (its field number)
- A narrative to be used as a column heading any input or output conversions required (such as for dates) the data type (left or right justified, or text that will break on word boundaries)
- A column width, used in the reports
- Input and output conversion codes can also be used to manipulate the data by performing mathematical functions, concatenating fields, or by extracting specific data from the field.

Multivalued Files

JBASE uses a three-dimensional file structure called a non-first normal form data model to store multiple values for a field in a single record known as multivalued fields. A multivalued field holds data that would otherwise be scattered among several interrelated files. Two or more multivalued fields can be associated with each other when defined in the file dictionary. Such associations are useful in situations where a group of multivalued fields forms an array or are a nested table within a file. You can define multivalued fields as belonging to associations in which the first value in one multivalued field relates to the first value in each of the other multivalued fields in the association, the second value relates to all the other second values. Each multivalue field can be further divided into subvalues, again obeying any relationships between fields.

Chapter 2 Sentence Construction

A jQL Command sentence must contain at least a Command and a File name. The Command specifies which process to perform and the filename indicates the initial data source.
You can add optional clauses to refine the basic Command. You can use clauses to control the range of eligible record keys, define selection and sorting criteria, or to specify the format of the output, and so on.
REMEMBER: only a verb and filename are required. The following list summarizes each element in the Syntax.

Element	Description
Verb	Specifies the action to be performed
DICT	Queries the dictionary of the file you specify in filename instead of the data file. If specified, DICT must precede filenames in the query.
filename	The name of the file. Filename is required
records	A list of record ID's which specify the records on which the query operates. Enclose the record IDs in double quotation marks.
FROM *list#*	A number from 0 through 10 of an active select list that contains record IDs. The query operates on those records whose record IDs are in the select list.
Selection-expression	Specifies the conditions that data in a record must meet in order for the record to be selected for the query
sort-expression	Specifies the record list order.
output-specification	Specifies either the names of the fields for output or one or more EVAL expressions. An output expression can also include special keywords that direct the processing of field values for output
output-limiter	The **WHEN** clause, used to limit the output of multivalued fields
report-qualifiers	Special keywords used in formatting reports

JQL VERBS

Verb	Description
BSELECT	Retrieves selected records and generates a list composed of data fields from those records as specified by any explicit or default output specifications. Each subvalue within a field becomes a separate entry within the list.
COUNT	Counts the records in a file
ESEARCH	Similar to SEARCH
LIST	Displays data from records in a file
LIST-ITEM	Displays full listing of selected records
LIST-LABEL	Displays records in a format suitable for mailing labels and other block listings
REFORMAT	Redirects jQL output to a file or tape.
SEARCH	Creates a select list of records that contain an occurrence of one or more specified strings
SELECT	Creates a list of records that meet specified selection criteria
SORT	Lists selected records in sorted order
SORT-ITEM	Displays full listings of selected records in sorted order
SORT-LABEL	Displays items in a format suitable or mailing labels and other block listings
SREFORMAT	Redirects jQL output to a file or to a tape with records sorted by sort expression
SSELECT	Creates a sorted list of records that meet specified selection criteria
STAT	Displays numeric statistics for fields in a file
SUM	Adds numeric values in fields of records that meet specified selection criteria
T-DUMP	Copies records from disk to tape
T-LOAD	Copies records from tape to disk

COMMAND SYNTAX

jQL-Command file-specifier {record-list} {selection-criteria} {sort-criteria} {USING file- specifier} {output-specification} {format-specification} {(options}

SYNTAX ELEMENTS

JQL Command	One of the verbs like Commands detailed later. Most Commands will accept any or all of the optional clauses.
File specifier	Identifies the main data file to be processed. Usually the data section of a file, but could be a dictionary or a secondary data area.
Record list	Defines which records will be eligible for processing. Comprises an explicit list of record keys or record selection clauses. An explicit list comprises one or more record keys enclosed in single or double quotes. A selection clause uses value strings enclosed in single or double quotes and has at least one relational operator. If no record list is supplied, all records in the file will be eligible for processing unless an “implicit” record list is provided by preceding the Command with a selection Command such as GET-LIST or SELECT.
Selection criteria	Qualify the records to be processed. Comprises a selection connective (WITH or IF) followed by a field name. Field names can be followed by relational operators and value strings enclosed in double quotes. Logical Connectives AND/OR ca also be used. Expressions can be parenthesized to specify procedure.
Sort criteria	Specify the data list order. Comprises a sort modifier, such as BY or BY-DSND, followed by a field name. Used also to “explode” a report by sorting lines corresponding to multivalued fields by value, and to limit the output of values (see output specification).
USING file specifier	Defines an alternate file for use as the dictionary.
Output Specification	Comprises the names of the fields to be included in the report, optionally preceded by a BREAK-ONconnective or ‘TOTAL’ connective. Print limiters (Values strings enclosed in double quotes after the field name, optionally preceded by relational operators) can be used to restrict multivalue output
Format specification	Comprise modifiers, such as HEADING, ID-SUPP, and DBL-SPC, which define the overall format of the report.
options	Comprises letters enclosed in parentheses, which modify the action of the Command to redirect output to a printer for Example.

Comments: A macro can substitute any element of a jQL Command sentence (with the exception of the Command and filename). When the REQUIRE-SELECT modifier is included in a jQL sentence it verifies that a select list is active before processing the sentence.

Macros

jQL allows the use of macros to predefine parts of a sentence. The macro definition contains one or more optional sentence elements. You invoke the macro by including its name in a sentence. The jQL processor looks for the macro in the currently active dictionary and includes all of its text parts in the sentence.

Using Clause

You may include multiple USING clauses in the sentence to specify that another file contains the attribute and the macro definition items referenced in the sentence. Each macro definition can also contain a USING clause to define the terms used within that macro.

Options

Options are letters enclosed within parentheses, which modify the action of the verb

Print Limiters

Sort criteria and format specifications can include ‘print limiters’ which suppress output of particular values according to specified criteria.

Record IDs

jBASE treats record ID’s differently from other fields in a file, in that the record ID is the default field displayed by LIST/SORT commands and the implied field for selection clauses.

The @ID Field

The @ID Field is optional in jBASE files. The field name is @ID; the column name is set automatically to the name of the file, and the output format defaults to 10L

The @ID Synonym

The @ID synonym is an optional entry you may find in some file dictionaries particularly in table dictionaries or in jBASE files converted from CONVERT.SQL utility. Under prime emulation, @ID contains the default output specification of @LPTR for output to printers.

Entering a jQL Command Sentence

A jQL Command sentence is entered at the shell in response to a Command prompt (:) or a select prompt (>). If a Command such as SELECT or GET-LIST creates an implicit list whilst in jSHELL, it displays the select prompt. Each sentence must start with a jQL Command and can be of any length. Press <ENTER> to submit the constructed sentence. If you enter an invalid Command, the system will reject it and display an appropriate error message.

EXAMPLE

```
SORT SALES WITH PART.NO = "ABC]" BY POSTCODE CUST.NAME POSTCODE
TOTAL VALUE DBL-SPC HDR-SUPP (P
```

SORT the jQL Command.
SALES the filename
WITH PART.NO = "ABC]"

The selection criterion: select all records, which contain a part number beginning with ABC.
BY POSTCODE
The sort criterion
CUST.NAME POSTCODE TOTAL VALUE

The output specification:

a. Column 1 will contain the key of the SALES file
b. Column 2 will contain the customer name
c. Column 3 will contain the POSTCODE.
d. Column 4 will contain VALUE (this is totalled at the end of the report)

DBL-SPC HDR-SUPP

The format specifications - Double-space the lines and suppress the automatic header.
(P
An option: redirect output to the system printer, rather than to the terminal.
PART.NO, CUST.NAME, POSTCODE, VALUE

References to data definition records defined in the dictionary level of the SALES file

GENERAL RULES FOR JQL SENTENCES

Entering a Sentence

The sentence is entered at the TCL prompt (:) or the select prompt (>) in an account which contains a pointer to the referenced file. If a verb such as SELECT or GET-LIST supplies an item, it displays the select prompt.

The following rules apply when entering a sentence:

1. The sentence is entered on one command line and is completed when selecting <ENTER>
2. It must begin with a verb, defined in the MASTER DICTIONARY followed by a space.
3. It must contain a file name, which is defined within the account from, which the sentence is entered
4. Unless otherwise stated, one or more spaces should separate each element.

Line Continuation

When you are typing words in response to the TCL prompt the system allows you to enter up to 240 characters before it performs an automatic linefeed. You can extend a line by entering the line continuation characters.

To enter the continuation sequence hold down the CTRL key and press the underscore key (_), which may require you to hold down the shift key. Follow this combination immediately with the RETURN key.

Procedure

1. Before the end of the 240 characters, enter CTRL+- and then press RETURN
2. Type the next part of the sentence. If a space is required at this point be sure to press the space bar either before or after step
3. If more than 240 characters are required repeat step
4. At the completion of the sentence press RETURN.

EXAMPLE

If a sentence exceeds 240 characters (see below), the line continuation sequence (CTRL+- and RETURN) breaks the sentence into two parts.

```
LIST GUEST WITH INVOICE-CODE "15" AND WITH EACH INVOICE-CODE # "2"
BREAK-ON LAST-NAME TOTAL INVOICE CTRL+- RETURN
```

```
GRAND-TOAL "AMOUNT DUE 'U' "DET-SUPP HDR-SUPP RETURN
```

Action by the System

Upon completion of the sentence, the system checks with the MASTER dictionary for a definition of the verb, which if defined, the system checks the file name and the syntax of the sentence. If you enter an invalid verb or file name or the syntax is incorrect, or have exceeded the limitations the system displays an error message.

Generating jQL Reports

You can display jQL reports at a terminal or send to a printer by using the LIST or SORT verb (and other verbs). The default output device receives the reports, which could be a printer (or other currently assigned spooler device) by adding LPTR or option p.

Columnar or Non-columnar format

If the report will fit within the page width of the output device, it is output as a series of columns. However, if the report would require more than the current page width to be output in columns, it is output as a series of lines (known as non-columnar format).

Paging of report

If the report is displayed at the terminal and extends over more than one screen, press <ENTER> to view the next screen.

Use the following words and symbols as described in this manual as all have special significance within a jQL sentence. These words are defined in each Master Dictionary (MD) and their definitions should not be changed in any way.		
!	#	&
<	<=	=
=<	=>	>
>=		
A	AFTER	AN
AND	ARE	

BEFORE	BETWEEN	BREAK-ON
BSELECT	BY	BY-DSND
BY-EXP	BY-EXP-DSND	
CAPTION	CHECK-SUM	COL-HDR-SUPP
COL-SPACES	COUNT	
DATA	DEL-SPC	DET-SUPP
DICT		
EACH	EDELETE	EQ
ESEARCH	EVERY	
FILE	FOOTING	FOR
GE	GRAND-TOTAL	GT
HASH-TEST	HDR-SUPP	HEADER
HEADING	I-DUMP	ID-SUPP
IF	IN	ISTAT
ITEMS	LE	LIST
LIST-ITEM	LIST-LABEL	LPTR
LT		
NE	NO	NOPAGE
NOT		
OF	ONLY	OR
PAGE	PG	REFORMAT
S-DUMP	SELECTSORT	SORT-ITEM
SORT-LABEL	SREFORMAT	SSELECT
ST-DUMP	STAT	SUBVALUE

SUM	SUPP	T-DUMP
T-LOAD	TAPE	THE
TOTAL	USING	VALUE
WITH	WITHIN	WITHOUT

Predefined words and Symbols

jQL Verbs Overview

A jQL verb is the first word in the sentence and is the name of a processor that performs the required retrieval and manipulation. The Master Dictionary defines the names of the processors in your account's..

Frequently used Verbs

The most commonly used jQL verbs are:

LIST	SELECT	SORT	SSELECT

Other jQL Verbs include

BSELECT	COUNT	EDELETE
ESEARCH	I-DUMP	LIST-ITEM
LIST-LABEL	REFORMAT	SORT-LABEL
SORT.ITEM	SREFORMAT	ST-DUMP
T-DUMP	T-LOAD	

With the exception of EDELETE and T-LOAD, jQL verbs do not affect the contents of the original file(s). REFORMAT and SREFORMAT write to a nominated file or tape.

Generating Implicit Lists

The following generate an implicit list

BSELECT GET-LIST ESEARCH

SEARCH QSELECT SELECT SSELECT

File Specifiers and Modifiers

The filename implies two logical files:

1. A dictionary section

The dictionary section contains definition terms,

   a. Such as data section definition terms
   b. Data definition items (known as attribute definitions items)
   c. And macro definition items.

2. An enclosed data section.

The retrieved or referenced data is contained within the data section.

File Modifiers

As described below file modifiers DICT, ONLY, WITHIN and TAPE modifies the use of the file, and how it is accessed

SYNTAX

```
{DICT} {ONLY} {WITHIN} {TAPE} filename{,data-section-name}
```

SYNTAX ELEMENTS

DICT	Specifies the dictionary section of the file and contains the data for referencing. You must type the modifier DICT before the filename. When modifying a *filename* by the DICT the processor looks in the MD for attribute and macro definition items.
ONLY	Specifies that only item-ids are to be output and suppress any default output contents. You can type the modifier ONLY before *filename* or following all clauses, which contain attribute names.
WITHIN	Specifies a sublist such as bill of material items. Use WITHIN only with the LIST and COUNT verbs and must precede filename. Specify one item-id only; if you enter more than one item-id, it displays an error message.
TAPE	Tells the processor to retrieve data from a magnetic tape, which written only in a T-DUMP format. This modifier cannot be used with the sorting verbs such as SORT and ST-DUMP, nor with tape output verbs, such as T-DUMP, nor with the updating verb EDELETE
filename	Specifies a dictionary section and a data section
data-section-name	Specifies a data section other than the data section called filename. It must follow filename and use a comma with no spaces for separation.

ITEM LISTS

Introduction

An item list specifies the items within the file to be further processed. If no list is given, all items in the file are implied.

Types of Item List

An item list takes one of three forms:

a. An explicit item-id list
b. An implicit item-id list
c. An item-id selection clause

You cab combine Item-id selection with ***implicit*** but not with ***explicit*** item-id lists. You can combine every type of list with selection criteria based on attribute values.

Explicit Item-id list

An explicit item-id lists items for processing, which encloses each item-id in double quotes. Spaces between item-ids are optional. An item-id list cannot include a relational operator and ignores any included logical connectives.
JQL treats the values you place between quotes as item-ids, not as value strings. This treats the left ignore, right ignore and wild card as ordinary characters and not as special characters.

SYNTAX

```
'item-id' {'item-id'}.. .
```

EXAMPLES

Implicit Item-id list

To provide an implicit item-id list execute a verb such as SELECT or GET-LIST immediately before executing a jQL command. If you also specify item-id selection, the jQL processor effectively ANDs its result with the implicit item-id list to limit further the items selected.
If you specify an explicit item-id list, the processor ignores any implicit list.

Item-id Selection clause

An item-id selection clause expresses limits on the value of item-ids, for selection for processing. It has at least one value string that defines an item-id or part of an item-id, and an explicit relational operator must precede at least one value string. The relational operator is what makes jQL treat item-id selection differently from an explicit item list. You can use logical connectives to combine relational operations. If you do not use an explicit logical connective, jQL defaults to the OR connective. JQL searches the file for each item-id that matches the value strings in the criteria. If an implicit item-id list has been specified, the processor checks only those item-ids present in the list.

SYNTAX

```
{'value string'}. . . relational operator 'value string' {{logical-
connective} {relational operator} 'value string'}...
```

Value Strings

Value strings are character strings enclosed in delimiters (usually single quotes within item-id-selection criteria and double quotes within ordinary selection criteria); also used to compare against character strings in the file. The value string cannot contain the character by which it is delimited. For example: if the value string is enclosed in single quotes, it may contain double quotes, but not single quotes. Otherwise, the value string can contain any printable character, excluding RETURN, LINE FEED, and system delimiters. The simplest value string is a character string that has precisely those characters for testing (for example. 'Johansen') however a value string can also include the following special characters:
Left ignore ([) at the beginning of the string to indicate that the item-id may start with any characters (for example,'[son')

Right ignore (]) at the end to indicate that the item-id may end with any characters (for example, Johan]')

Wild cards(^) anywhere within the string, each wild card matching one character of any value (for example, 'Joh^ns^n).

Single and Double Quotes

Values string delimiters are single quote (') and double quote ("). You can enclose an item-id value string in double quotes, but only if it is entered immediately after the file name. Use single quotes

within item-id selection clauses and doubles quotes within ordinary selection criteria except when you are searching for an item-id that includes single quotes.

Between Connective

The connective BETWEEN followed by two value strings is a shorthand way of saying ‘all values greater than the first value string and less than the second’. The value of the second value string must be greater than the value of the first to select items. Value strings including special characters ^, [and] are not valid.

Relational Operators

These express a relationship between an item-id (or attribute value in the case of selection criteria) and the value string. At least one relational operator is required in an item-id selection clause. Value strings within the clause not preceded by a relational operator are treated as if preceded by the equal operator.

The operator test for relationships Equal(=), less than or equal (<=) etc., the result of a relational operation is a truth-value: true or false. You can enter relational operators as special characters symbols or as their mnemonic equivalents:

Data Type	Relational Operator	Synonyms	Description
Numeric Fields	EQ	=	Equal
	NE NOT NO	#	Not Equal
	GT	> AFTER	Greater Than After
	GE	>=, =>	Greater than or equal
	LT	<, BEFORE	Less than Before
	LE	<=, =<	Less than or equal to

String Fields	LIKE	MATCHES MATCHING	Matches a pattern or text
	UNLIKE	NOT.MATCHING	Does not match a pattern or text
	SAID	SPOKEN, ~	Sounds like
	EQ	=	Equal to
	NE	#	
	GE	>=, =>	Greater than or equal to
	GT	>, AFTER	Greater Than
	LE	=<, >=	Less than or equal to
Null Values	IS.NULL		Tests if a field for a null value
	IS.NOT.NULL		Tests a field for no null values

Logical Connectives

The logical connective **AND** or **OR** joins two relational expressions. The default connective is **OR**. If giving two relational expressions without a logical operator between them, items satisfying either expression are selected (as if the OR connective had been used).

The connective AND yields a truth-value of true if all the truth values it is combining are true. If any truth-value is false, the result of the AND connective is false. The OR connective yields a truth value of true if at least one of the truth values it is combining is true.

Synonyms

Ampersand (&) is a synonym for AND

Exclamation point is a synonym for OR

EXAMPLES

Item-id List

The following sentence lists information about rooms 117 and 119. Because there is no explicit relational operator this is an item list, and the processor accesses the items directly

```
LIST ROOMS '117' '119'
```

Value String Example

The following sentence lists information about rooms with numbers matching "117" or "119". Note: the equal sign makes these values strings rather than item-ids. Hence, without an implicit item list, the processor must search the entire file, comparing all items-ids against these two value strings; thus it would be better to omit the equal sign as shown in the previous example, to avoid this;

```
LIST ROOMS = '117' '119'
```

Implicit List Examples

The following sentences will not list anything because the value strings cannot match any item-id in the implicit list.

```
SELECT ROOMS GT '200'

23 ITEMS SELECTED

>LIST ROOMS = '117' '119'
```

The following sentences list information about rooms 117 and 119 because the process ignores an implicit item-id list when an implicit item-id list is in the sentence.

```
:select rooms GT '200'
```

```
23 items selected

>list rooms '117' '119'
```

Left Ignore examples

The following sentence lists information about all the rooms with numbers ending in 23.

```
LIST ROOMS = '[23'
```

The following sentence does not list any rooms because there is no relational operator, the value [23 is treated as an item-id.

```
LIST ROOMS '[23'
```

Wild Card Examples

The following sentence list information about all the rooms with numbers that begin with three, end with five, and have an intervening character of any value

```
: LIST ROOMS = '3^5'
```

The following sentence does not list any rooms because there is no relational operator, the string 3^5 is treated as an item-id.

```
LIST ROOMS '3^5'
```

AND Connective Examples

The following sentence lists information about all the rooms numbered 200 to 399.

```
:LIST ROOMS => '200' AND < '400'
```

The following sentence results in a report listing information only about the room 119 because in the absence of a relational operator, assumes an equal (=). The only room number greater than 117 and equal to 119 is 119.

```
:LIST ROOMS > '117' AND '119'
```

Apparent item-id List Example

The following sentences do not list information regarding 117 and 119 because they would not be on the implicit list. Although this sentence seems to have an explicit item-id list and an item-id selection clause, the whole series is a selection clause because there is a relational operator somewhere in the list.

```
:SELECT ROOMS GT '500'
```

```
4 items selected
>LIST ROOMS '117' '119' OR = '[27'
```

Further Examples of Item Lists

a. The following sentence lists information about rooms with numbers that are both greater than or equal to 400 and less than 700:

```
LIST ROOMS >= '400' AND LT '700'
```

b. The following sentence displays information about rooms with numbers less than 200 and with available dates after May 17 2002.

```
LIST ROOMS < '200' WITH AVAILABLE AFTER "MAY 17 2002"
```

The following sentence displays information about rooms with numbers less than 500 and greater than 199 and with bed codes to either suite or double. The second AND arises because the sentence includes both item selection and data selection criteria: these operations perform one after the other, giving an effective AND function. The OR between "ST" and "D" is implicit.

```
LIST ROOMS LT '500' AND GT '119' WITH ROOM CODE "ST" "D"
```

c. The following sentence lists rooms with numbers less than 200 or greater than 399.

```
LIST ROOMS < '200' OR > '399'
```

Selection Criteria Clause

The selection criteria clause allows you to specify data-specific limits on the range of records that will be eligible for processing.

If a record list of any type is outstanding when processing reaches the selection criteria, only those in the list will be submitted to the selection process; if there are no record lists outstanding the selection process considers all records in the file.

Each selection criterion specifies a field (data or key) for testing to determine selection of a record. The selection criterion begins with the connective (WITH (or IF) and must also include a field name. The field name can be followed by a value selection clause otherwise it defaults to NE ""(not equal NULL)

SYNTAX

```
WITH | IF { NOT } { EACH } field {value-selection clause} {{AND |
OR}
{ WITH |IF } {NOT} {EACH} field {value-election clause}...}
```

Value selection clause has the form:

```
{relational-operator} "value string" {{logical-connective}
{relational operator} "value string"}. . .
```

SYNTAX ELEMENTS

WITH or IF is the selection connective. It must be the first word of a selection criterion. WITH and IF are synonymous. WITHOUT is a synonym for WITH NOT.

Sort Criteria Clause

The sort criteria clause allows you to specify the presentation order of the records in the report.

SYNTAX

```
BY field
BY-DSND field
BY-EXP field {print-limiter}
BY-EXP-DSND field {print limiter}
```

Print limiter has the form:

```
{relational operator} "value string" {{logical connective}
{relational operator} "value string"}...
```

field is the name of a data definition records.

BY	Specifies a single value sort that will order the records according to an ascending sequence based on the first value in the specified field
BY-DSND	Specifies a single value sort the which will order the records according to a descending sequence based on the first value in the specified field
BY-EXP	Specifies a multivalue sort that will order the multivalues of the specified field

	according to an ascending sequence based on the first subvalue in each multivalues element
BY-EXP-DSND	Specifies a multivalues sort that will order the multivalues of the specified field according to a descending sequence based on the first subvalue in each multivalued element

Comments: Each sort clause comprises a sort connective followed by a field name. The sort connective can specify an ascending or descending sort sequence of single or multivalued fields. If you include more than one sort of criteria clause, the processor ranks the clauses in a left to right, most to least important hierarchical sequence. Always used as the least important sort value, unless explicitly included in the SORT criteria is the record key.

The precise sorting sequence depends on whether a field is left - right justified.

Default sort order

If you do not specify a sort criteria clause for a sorting command, the report is output in ascending order by record key. Field 9 of the file definition (pointer) record specifies Left or right justification of the key. The default is a left justified sort.

Sort Order of Left justified Data

When sorting a left justified field the data is compared one character at a time, left to right. For this reason number two will follow number 11 in an ascending sequence. The number 02 would appear before 11 in an ascending sequence.

The sort connectives for single valued fields sort the record orders according to the value of a field.

The two sort connectives for single value fields are:

1. BY for an ascending sort
2. BY-DSND for a descending sort

If using a single value sort connective with a field that contains multivalues or subvalues, it only uses the first value in the field as the sort key.

Sorting Multivalued Fields

The sort connectives for multivalued fields sort values within a field.

The two sort connectives for multivalues are:

```
BY-EXP for ascending order
```

```
BY-EXP-DSND for descending order
```

If using a multiple value sort connective with a file, which contains subvalues, it only uses the first subvalue in each multivalue as the sort key.
The treatment of each value is as if it were the only value so that each value occupies a line of output in the report. This effectively "explodes" a record into multiple records.
You can limit the values for sorting and output by including a print limiter with the multivalue sort connectives.
When using a SELECT-type command with BY-EXP the formatting of the records list appears: record-key]multi value#

Where: multi-value-# is the position of the multivalue within the field. The READNEXT statement in a jBASIC program can use this value.

EXAMPLE 1

```
SORT SALES WITH S.CODE = "ABC]" BY S.CODE
```

Selects the records in the SALES file in which the S.code field contains values, which start with ABC. The output in the records is in S.CODE order.

EXAMPLE 2

```
SORT SALES WITH S.CODE = "ABC]" BY S.CODE BY-DSND VALUE
```

EXAMPLE 3

```
SORT SALES BY-EXP P.CODE
```

Selects all the records in the SALES file and outputs the detail lines in key order within P.CODE order.

Using Clause

Comprises {DICT} *filename{data-section-name}*, where DICT specifies the dictionary *filename, filename* specifies a file, *data-section-name* specifies a secondary data section of *filename.*

Output Specification Clause

The output specification clause names the fields that are to be included in the report.

SYNTAX

```
field {print limiter}
TOTAL field {print-limiter}
BREAK-ON field "{text}{2option{option}...'}{text}
Print limiter has the form:

{NOT} {relational operator} "value string" {{logical-connective}
{NOT} {relational-operator} "value string"}...
```

SYNTAX ELEMENTS

TOTAL specifies that a running total of a numeric field be maintained

field identifies the name of a data definition record

Print limiter suppresses output of values (to subvalue level) that do not match the clause, which replaces suppressed values with blanks. Any detail lines that would as a result, be blank, are suppressed. Any totals produced include just the values that match the limiting clause.

BREAK-ON specifies that control break be performed and a break line displayed, each time the value of a field changes

Text comprises any printable characters except **RETURN**, **LINE FEED**, double quotes, single quotes or system delimiters.

Options is one of more of the following options:

B	Break: works in conjunction with the B option of the Heading and FOOTING modifiers to insert the break value in the heading or footing.
D	Data: suppresses the break if only one detail line has been output since the last break.
L	Line: suppresses the blankline preceding the break data line. Overrides the U option if both are specified.
P	Page: throws a new page after each new break value until all the data associated with the current break has been output.
R	Rollover: Inhibits a page break until all the data associated with the current break has been output.
U	Underlines: if specified places underlines on the line above the accumulated totals. Ignored if used with the L option.

V	Value: inserts the value of the control break field at this point in the BREAK-ON option.

Comments: If the sentence contains an output specification clause, it ignores any default definition records in the dictionary.

Total Connectives

The TOTAL connective specifies that a running total of the field be maintained and to output the total at each control break and at the end of the report. Also, use TOTAL in conjunction with the BREAK-ON connective to display intermediate totals at the control breaks.
Use the GRAND-TOTAL modifier in the format specification clause to display any specified text on the last total line.

BREAK-ON Connective

The BREAK-ON connective causes monitoring of the following fields for change permitting up to fifteen breaks within one sentence treated in hierarchical left to right order. The first **BREAK-ON** in the sentence is the highest level.
When detected, the change in the value of the field outputs a blank line, followed by a line with three asterisks, and then another blank line. If the BREAK-ON clause specifies text, it outputs the text in place of asterisks. If the text is wider than the column width, the processor applies the same justification as the named field.
You can suppress the BREAK-ON output by setting the column width of the field to zero.
You can use BREAK-ON in conjunction with the TOTAL connective to generate subtotals. If using the modifier DET-SUPP with TOTAL and BREAK-ON, it displays only the subtotal and grand total lines.

BREAK-ON Options	
B	Break. Works in conjunction with the B option of the heading and footing modifiers to put the break values in the heading or footing
D	Data. Suppresses the break line if there is only one detail since the last BREAK. This is the line with the asterisks, any text that is specified, or totals
L	Line. Suppresses the blank line preceding the break data lie. Overrides the U option if both are specified
P	Page. Causes each break item to be output on a separate page

R	Rollover. Inhibits a page break until all the data associated with the current break is output
U	Underlines. Places underlines on the line above the accumulated totals if the TOTAL modifier was specified. Ignored if used with the 'L' option
V	Value. Causes the values of the control Break attribute to be inserted at this point in the BREAK-ON label

Controlling and Dependent Fields

Controlling and dependent fields provide a method for creating sublists from records.
A controlling field is one, which has the code D1 in field 8 of its data definition record and points to its controlling field.

When the system finds a controlling field, it will:

a. Look for the first field specified in the output specification clause that matches each FMC (Field Mark Count) of its dependent field and has D2 code in field 8 of the data definition item specifying the controlling field.
b. Position the found fields in the order found to the immediate right of the controlling field for display.
c. Display an asterisk (*) under the column heading of each found field.
d. Dependent fields are output immediately to the right of their controlling field regardless of the order in which you specify them.
e. An independent field found between the controlling and dependent fields is moved "logically" to the right of the controlling and dependent fields.
f. Will ignore dependent fields unless you specify the controlling field.

EXAMPLE 1

```
SORT SALES P.CODE S.CODE ="ABC"
```

Selects all the records in the SALES file and outputs the P.CODE data. The S.CODE data will only be included if it matched ABC - any other value will be shown as blank.

EXAMPLE 2

```
SORT SALES BY P.CODE BREAK-ON P.CODE TOTAL VALUE
```

Selects all the records in the SALES file in P.CODE order and outputs a line for each record until the P.CODE changes. At this point, a control break triggers and outputs the running total of VALUE. At the end of the report, it displays a cumulative total for VALUE.

Formatting Reports with Report Qualifiers

Using report qualifiers, you can tailor the layout of the entire report by setting up headers and footers on each page, adjusting margins and spacing, and determining output orientation (horizontal or vertical). In addition, there are two jQL commands, LIST-LABEL and SORT-LABEL, which enable you to format and sort mailing labels.

Using Report Qualifier Keywords

Report qualifiers provide a variety of ways to control and refine the overall format of a report. COL-HDG, ID-SUP, DET-SUP, LPTR, SAMPLE, and SAMPLED are report qualifiers you saw in previous examples. The following list summarizes the most commonly used report qualifiers:

Keyword	Synonym	Description
COL-HDR-SUPP	COL.HDR.SUPP	Suppresses the default report and column headings.
COL-SPCS	COL.SPCS	Changes the default spacing between columns.
COL-SPACES	COL.SPACES	Specifies the spacing between the columns of a report
COL-SUP	COL.SUP	Suppresses the column heading.
COUNT-SUPP	COUNT.SUPP	Suppresses the count displayed at the bottom of a report.
DBL-SPC	DBL.SPC	Double-spaces the report.
DET-SUPP	DET.SUP	Displays only the breakpoint lines.
DET-SUPP	DET.SUPP	Use with BREAK.ON.
FOOTING	FOOTER	Sets the report footing.
GRAND-TOTAL	GRAND.TOTAL	Sets the text for a grand total line.

HDR-SUP	HDR-SUPP, SUPP	Suppresses the default report heading.
HEADING	HEADER	Uses the report header you specify in the query rather than the default heading.
HEADING DEFAULT	HEADER DEFAULT	Displays the default heading.
ONLY	ID-ONLY	Displays record IDs only.
ID-SUP	ID.SUPP	Suppresses the display of record
LPTR	(P	Sends the output to the printer.
NOPAGE	(N	Specifies that the report is automatically scrolled on the terminal
SAMPLE FIRST		Displays the first n records
SAMPLED		DISPLAYS even nth record
VERTICALLY	VERT	Displays the report in vertical format with one field on each line.

B	Functions only if a BREAK-ON modifier with a B option is also included in the sentence. You can use the B option in either the header or footer. When the B Option is in the HEADING the value of the first BREAK-ON field on the page replaces the B in the header. When the B is in the FOOTING, the last BREAK-ON value on the page replaces the B in the footer.
C{n}	Centralizes the heading or footing text and centres the text according to the predefined number of the columns specified for the printer or terminal. To change the centering of the text specify the number of columns (n) for the heading line on which to base the center. For example: ‘C80’ positions the text centered at character position 40. You should allow the printer or terminal set-up to determine the centering.
D	Inserts the current date using the format: dd mmm yyyy.
F	Inserts the file name
I	Inserts the current record key. The last record key listed on the page is inserted in the footing; the first key on the page is inserted in the heading

L	Specifies that a new line is to start where the L appears
N	Specifies suppression of automatic paging.
P	Inserts the current page number right justified, expanding to the right as the number increases.
PP	Inserts the current page number right justified in a field of four spaces
T	Inserts the current system time and date in the format: hh:mm:ss dd mmm yyyy

GRAND-TOTAL

Specifies the text to replace the default asterisks in the cumulative total line at the end of the report; CAPTION is a synonym for GRAND-TOTAL.

L	Line: suppresses the blank line preceding the GRAND-TOTAL line. Overrides the U option if both are specified and
P	Page: outputs the GRAND-TOTAL on a separate page.
U	Underline: places underlines on the line above the accumulated totals. Ignored if used with the 'L' option.
LPTR	Specifies that a report go to the printer queue (spooler) instead of displaying at the terminal. You could use the 'P' option at the end of the sentence in place of this modifier.

Comments: Enter a heading or footing option, which specify a value in the order in which they appear.

Text spaces are not normally required between option codes. However, you can present options that represent values such as pages or dates without spaces. For example: '"PD"' will print on the first page as:
111/11/00
In this case, enter the options with a space between them like this "'P' 'D'"

EXAMPLE

```
SORT SALES BY S. CODE BREAK-ON S.CODE "'BL' "P.CODE TOTAL VALUE
GRAND-TOTAL "Total "HEADING "SALES CODE : 'B' 'DL'" FOOTING "PAGE
```

```
'CPP' "LPTR
```

Control Break on a change in S.CODE and suppress the LINE FEED before the break. Reserve the break value for use in the heading ('B')
Maintain a running total of the VALUE field and output it at each control break.
Put the word Total on the GRAND-TOTAL line.
Set up a heading for each page, which comprises the words 'SALES CODE :', the sales code (from the break), a date and a line feed. Set up a footing, this contains the text 'PAGE', and a page number, centered on the line.
Produce the report on the currently assigned printer.

Using Clause

The using clause specifies a dictionary file, which is the source of data definition records.

SYNTAX

```
USING {DICT} filename {,data-section-name}
```

SYNTAX ELEMENTS

USING specifies the use of the named file as the dictionary for the data file.
DICT specifies that the dictionary of filenames be used
filename names a file. If the DICT modifier is not specified, it will use the data section of the file.
data-section-name specifies a secondary data section of the file with a name different from the dictionary; it must follow filename, separated by a comma but no space.
Comments: You can use as many USING clauses required in a jQL sentence.
One main advantage of the using clause is that you can share a dictionary between several files where for example there are common data definition records.

EXAMPLE

```
SORT SALES.94 USING DICT SALES
```

The data definition records in the dictionary of the file sales (DICT SALES) assess File SALES .94

Command Options

Command options are letters enclosed in parentheses, which modify the action of the jQL command sentence.
The options described here are common to most commands. Where the options are command-specific, they are described with the command.
Do not confuse options for commands with options for modifiers and connectives such as **HEADING** and **BREAK-ON**.
Commas or spaces can separate options; when the options are at the end of the sentence (as is recommended) omit the closing parenthesis.
Ignores any option, not used by a particular command

Options	
B	Suppress initial LINEFEED prior to out put
C	Suppresses column headings, page and date, line at the start and summary line at the end of a report: Equivalent to the COL-HDR-SUPP modifier
D	Suppress detail output equivalent to the DET-SUPP modifier
H	Suppress page and date line at the start and summary line at the end of the report: Equivalent to HDR-SUPP modifier
I	Suppress record keys: equivalent to the ID-SUPP modifier
N	Suppress automatic paging: equivalent to the NOPAGE modifier.
P	Output report to the printer: equivalent to the LPTR modifier
S	Suppress summary line at the end of the report

EXAMPLE

```
LIST SALES (HIP
```

List the SALES file (using the default data definition records) but suppress the output of a header and the record keys. Send the output to the assigned printer.

Macros

Macros contain predefined or often used elements of a jQL sentence, stored on the system like data definition records and are specified in the command sentence in a similar way.
When submitting a command containing one or more macros for execution it expands and includes the macro references in the sentence.
You can substitute macros for any element of the command sentence except the command itself and the filename.
The search for macro definition records is in the same way as data definition records. Do not use a jQL keyword for a Data Definition record.
The first field of a macro definition must contain the letter M. The remaining fields are either command elements or comment lines (indicated by a leading asterisk '*' and a space).
You can nest macros - a macro can refer to another macro - but the resulting command sentence must still follow the same rules as a normal jQL sentence. When nesting macros, beware of infinite loops where for example, macro A calls macro B that calls macro A that calls macro B.

EXAMPLE

```
SORT SALES BY S.CODE STD.HEADING
```

In this example, STD.Heading is a macro, which contains a standard heading clause:
STD.HEADING

```
001 M

002 * Standard heading for sales reports

003 Heading "SALES - COMPANY PRIVATE'LL'PAGE 'PL'"

004 LPTR
```

When the sentence expands it will look like this:

```
SORT SALES BY S.CODE HEADING "SALES - COMPANY PRIVATE'LL'PAGE '"
LPTR
```

Thowaway Connectives

Throwaway connectives are keywords, which make queries more readable. You can use in any query to make the sentence read more like English and can be used anywhere in a sentence as throwaway connectives do not affect the query.
The following query uses the words THE, FOR, and FILE without affecting the meaning of the command:

```
>LIST THE ITEM.CODE DESCRIPTION PRICE FOR THE INVENTORY.F FILE
```

The throwaway keywords are:

Throwaway Connectives		
A	ARE	FILE
FOR	INIVISIBLE	OF
PRINT	THAN	THE

COMMANDS

For example, entering the following command would be incorrect:

```
>LIST ITEM.CODE DESCRIPTION AND PRICE FOR THE INVENTORY.F FILE
```

Chapter 3 jQL Commands

BSELECT

Retrieves selected records and generates a list composed of data fields from those records as specified by any explicit or default output specifications. Each subvalue within a field becomes a separate entry within the list.

COMMAND SYNTAX

```
BSELECT file-specifier {record-list} {selection-criteria} {sort-
criteria} {USING file-specifier}{output-specification} {(options}
```

Comments: When the Command terminates, it displays the total number of entries in the generated list and makes the list available as if generated by a SELECT, GET-LIST or other list-providing Command.

If you do not specify a sort-criteria clause, the record list will be unsorted.

If you do not specify an output-specification, it uses the default data definitions “1”, “2” etc. .

EXAMPLE

```
BSELECT SALES WITH S.CODE = "ABC]" P.CODE
```

Creates a list containing all P.CODE values from all the records in the SALES file, which have an S.CODE that starts with ABC

COUNT

Reports the total number of records found in a file, which match the specified selection criteria.

COMMAND SYNTAX

```
COUNT file-specifier {record-list} {selection-criteria} {USING file-specifier} {(options}
```

SYNTAX ELEMENTS

Options can be one or more of the following:

Option	Description
B	Suppress initial line-feed.
C{n}	Display running counters of the number of records selected and records processed. Unless modified by n, the counter increments after every 500 records processed or the total number of records if less than 500. The n specifies a number other than 500 by which to increment. For Example, (C25) increments the counter after every 25 records processed.
P	Send the report to the printer.

EXAMPLES

```
COUNT SALES WITH VALUE > "1000"
```

91 Records counted

Count the number of records in the SALES file which have a value greater than 1000.

```
COUNT SALES WITH VALUE > "1000" (C50
```

91 Records selected 240 Records processed
91 Records counted

Count the number of records in the SALES file which have a VALUE greater than 1000, and display a running total of selected and processed records after each group of 50 records are processed.

EDELETE

Deletes selected records from a file according to record list or selection criteria clauses.

COMMAND SYNTAX

```
EDELETE file-specifier [record-list | selection-criteria]
```

Comments: EDELETE requires an implicit or explicit record list, or selection criteria. Preceding the Command with a SELECT, GET-LIST or other list-providing Command can provide an implicit list. EDELETE will immediately delete the specified records. To clear all the records in a file, use the CLEAR-FILE Command.

EXAMPLES

```
EDELETE SALES "ABC" "DEF"
```

2 Records deleted

Delete the records ABC and DEF based on the explicit list of records.

```
EDELETE SALES IF P.CODE = "GHI]"
```

n Records deleted

Delete all records in the SALES file in which the P.CODE field starts with GHI.

```
SELECT SALES WITH S.CODE = "ABC"
```

n Records selected

```
EDELETE SALES
```

n Records deleted

Selects all records in the SALES file in which the S.CODE field contains ABC, and deletes them.

ESEARCH

Generates an implicit list of records in a file if they contain (or do not contain) one or more occurrences of specified character strings

COMMAND SYNTAX

```
ESEARCH file-specifier {record-list} {selection-criteria} {sort-criteria} {USING file-specifier} {(options}
```

SYNTAX ELEMENTS

Options can be one or more of the following:

Option	Description
A	ANDs prompted strings together. Records must contain all specified strings
I	Displays the keys of selected records.
L	Saves the field numbers in which it found the specified strings. The resulting list contains the record keys followed by multivalued line numbers. Ignores the A and N options if either or both are specified.
N	Selects only those records that do not contain the specified string(s).
S	Suppresses the list but displays the record keys that would have been selected.

Prompt: At the prompt supply one or more search strings:
String: Enter the required character string and press <ENTER>. This prompt is repeated until only <ENTER> is pressed. You can enter unlimited characters.
Do not enter double quotes unless they are part of the string to search.
Comments: When the Command terminates (unless the "S" option is used), it displays the total number of entries in the generated list. The list is then available as if generated by a **SELECT**, **GET-LIST** or other list-providing Command. If you do not specify a sort criteria clause, the record list will be unsorted.

EXAMPLE

```
ESEARCH SALES (I
STRING: ABC
STRING: DEF
KEY1
```

```
KEY2

2 Records selected
>
```

Generates a list of all records in the SALES file, which contain the strings ABC or DEF

I-DUMP / S-DUMP

Displays the entire contents of items in a file, including the system delimiters

COMMAND SYNTAX

```
I-DUMP file-specifier {record-list} {selection-criteria} {sort-
criteria} {USING file-specifier} {(options}
```

Comments: Use the S-DUMP Command to produce sorted output.

Denoted as follows are system delimiters:

Attribute mark	^
Value mark	]
Sub value mark	\

EXAMPLE 1

```
I-DUMP EMPLOYEE WITH EMP.JOB = "SALESREP"
```

Generates the following output:

```
8499^HARRIS^TAMMY^SALESREP^8698^5400^250000^30000^90030^^^^^]11588^
8654^MCBRIDE^KEVIN^SALESREP^8698^5620^215000^140000^90030^^^^^]11639
\^
8521^TAYLOR^MAVIS^SALESREP^8698^5402^219000^50000^90030^^^^^3]11500^
```

EXAMPLE 2

```
S-DUMP EMPLOYEE BY EMP.HIREDATE WITH EMP.TITLE "SALESREP"
Generates the following output:
8499^HARRIS^TAMMY^SALESREP^8698^5400^250000^30000^90030^^^^^]11588^
8521^TAYLOR^MAVIS^SALESREP^8698^5402^219000^50000^90030^^^^^3]11500^
8654^MCBRIDE^KEVIN^SALESREP^8698^5620^215000^140000^90030^^^^^]1163\
^
```

LIST

Generates a formatted report of records and fields from a specified file

COMMAND SYNTAX

LIST file-specifier {record-list} {selection-criteria} {sort-criteria} {USING file-specifier} {output-specification} {format-specification} {(options}

Comments: If providing no output specification clause the system searches for default data definition records (named 1, 2 and so on) in the file dictionary and then in the file specified in the JEDIFILENAME_MD environment variable. If no default data definition records are found, it lists only the record keys. You must specify a sort criteria clause to sort the records.

EXAMPLE 1

```
LIST SALES
```

List all the records in the SALES file and use the default data definition records (if found) to format the output.

EXAMPLE 2

```
LIST SALES "ABC" "DEF" "GHI"
```

List the records from the SALES file with key values of ABC, DEF or GHI. Use the default data definition records (if found) to format the output.

EXAMPLE 3

```
GET-LIST SALES.Q4
>LIST SALES GT "DEF"
```

Get the previously saved list called SALES.Q4 and, using the list, report on the records in the SALES file which have a key greater than DEF. Use the default data definition records (if found) to format the output.

EXAMPLE 4

```
LIST SALES WITH S.CODE = "ABC]" OR "[DEF"
```

List the records in the SALES file in which the S.CODE field contains values which start with ABC or end with DEF. Use the default data definition records (if found) to format the output.

EXAMPLE 5

```
LIST SALES WITH NO S.CODE = "ABC]" OR "[DEF" (P
```

List the records in the SALES file in which the S.CODE field does not contain values which start with ABC or end with DEF. Output the report to the printer. Use the default data definition records (if found) to format the output.

EXAMPLE 6

```
LIST SALES BY S.CODE BREAK-ON S.CODE "”BL” P.CODE TOTAL VALUE GRAND-
TOTAL "Total” HEADING "Sales Code: "B” "DL” FOOTING "Page "CPP” LPTR
```

Sort the SALES file by S.CODE. Output the S.CODE, P.CODE and VALUE fields.
Control break on a change in S.CODE and suppress the LINE FEED before the break. Reserve the break value for use in the heading (“B”).
Maintain a running total of the VALUE field and output it at each control break.
Put the word “Total” on the grand-total line.
Set up a heading for each page which comprises the words “Sales Code: “, the sales code (from the break), a date and a LINE FEED. Set up a footing, which contains the text “Page”, and a page number, centered on the line?
Produce the report on the currently assigned printer.

LIST-LABEL

Outputs data in a format suitable for producing labels

COMMAND SYNTAX

```
LIST-LABEL file-specifier {record-list} {selection-criteria} {sort-criteria} {USING file-specifier}{output-specification} {format-specification} {(options}
```

Prompts

At the prompt, supply formatting criteria as follows:

COL, ROW, SKIP, INDENT, SIZE, SPACE{C}:

COL	The number of columns required to list the data across the page.
ROW	Number of lines for each record. Each element of the output specification appears on a separate line, if more elements exist in the output specification than rows specified it ignores the extra elements. if you specify more rows than elements, the output specification for these rows will be blank.
SKIP	Number of blank lines between each record.
INDENT	Number of spaces for left margin.
SIZE	Number of spaces required for the data under each column.
SPACE	Number of horizontal spaces to skip between columns
C	Optional. Suppresses null or missing data. If absent, outputs null or missing values as blanks. If present, the C must be upper case and not in quotes.

Comments: The total number of columns specified must not exceed the page width, based on the calculation:

COLs * (SIZE + SPACE) + INDENT <= page width

ROW must be a minimum of one for each field, plus one for the record key (if not suppressed). If the record keys are not suppressed, the first row of each label will contain the record key.

If **INDENT** is not zero, at the prompt supply a series of **HEADER**s that will appear in the left margin for each field. If a heading is not required for a particular line, press <ENTER>.

Multivalued fields appear as separate labels.

If specified, **COL-HDR-SUPP** or **HDR-SUPP**, or the **C** or **H** options, the page number, date, and time will not be output and generates the report without page breaks. You must specify a sort criteria clause to sort the records.
See also the **SORT-LABEL** Command.

EXAMPLE

```
LIST-LABEL SALES NAME ADDRESS STREET TOWN POSTCODE ID-SUPP (C
COL,ROW,SKIP,INDENT,SIZE,SPACE(,C): 2,5,2,0,25,4,C
```

NAME1	NAME2
ADDRESS1	ADDRESS2
STREET1	STREET2
TOWN1	TOWN2
POSTCODE1	POSTCODE2

NAME3	NAME4
ADDRESS3	ADDRESS4
STREET3	STREET4
TOWN3	TOWN4
POSTCODE3	POSTCODE4

LISTDICT

Generates a report of all data definition records in the first MD file found, or the specified file

COMMAND SYNTAX

```
LISTDICT {file-specifier}
```

SYNTAX ELEMENTS

file specifier - specifies a dictionary file other than a file named MD in the JEDIFILEPATH.

Comments: If you do not specify a file-name, LISTDICT will work with the first file named MD, it finds in your JEDIFILEPATH.

REFORMAT

REFORMAT is similar to the **LIST** Command in that it generates a formatted list of fields, but its output is directed to another file or the magnetic tape rather than to the terminal or printer.

COMMAND SYNTAX

```
REFORMAT file-specifier {record-list} {selection-criteria} {USING
file-specifier} {output-specification} {format-specification}
{(options}
```

Prompt

At the prompt, supply the destination file:

File: Enter a file name, or the word “TAPE” for output to a magnetic tape.

Comments: Overwrites records that already exist in the destination file; when you reformat one file into another, each selected record becomes a record in the new file. It uses the first value specified in the output specification clause as the key for the new records. The remaining values in the output specification clause become fields in the new records.

When you reformat a file to tape, it concatenates the values specified in the output specification clause to form one tape record for each selected record. The record output is truncated or padded at the end with nulls (X”00”) to obtain a record the same length as specified when the tape was assigned by the T-ATT Command.

Unless you specify HDR-SUPP or COL-HDR-SUPP, or a C or H option, a tape label containing the file name, tape record length (in hexadecimal), it will write the time, and date to the tape. If specifying a HEADING clause, this will form the data for the tape label.

Unless the ID-SUPP modifier or the 'I' option is specified record keys are displayed as the records are written to tape.

Two EOF marks terminate the file on tape.

See also the **SREFORMAT** Command.

EXAMPLE

```
REFORMAT SALES C.CODE NAME ADDRESS
FILE: ADDRESS
Creates new records in the ADDRESS file, keyed on C.CODE from the
SALES file. Each record contains two fields, one with the values
from the NAME field and one with the values from the ADDRESS field.
```

SELECT

Generates an implicit list of record keys or specified fields based on the specified selection criteria

COMMAND SYNTAX

```
SELECT file-specifier {record-list} {selection-criteria} {sort-
criteria} {output-criteria} {USING file-specifier} {(options}
```

SYNTAX ELEMENTS

The options are:

C{n} Display running counters of the number of records selected and records processed. Unless modified by n, the counter increments after every 500 records processed or the total number of records if less than 500.

n n Specifies a number other than 500 by which to increment. For Example, C25 increments the counter after every 25 records processed.

Comments: You must specify a sort criteria clause to sort records.

See also the **SSELECT** Command.

If you specify an output-criteria clause, the generated list will comprise the data (field) values defined by the clause, rather than the selected record keys.

If you are in jSHELL when the Command terminates, it displays the total number of entries in the generated list and the list is made available to the next Command, as indicated by the > prompt.

If you use the **BY-EXP** or **BY-EXP-DSND** connectives on a multivalued field, the list will have the format:

record-key]multivalue#

where multivalue# is the position of the multivalue within the field specified by **BY-EXP** or **BY-EXP-DSND**. multivalue# can be accessed by a **READNEXT** Var,n statement in a jBASIC program.

EXAMPLE 1

```
SELECT SALES WITH S.CODE = "ABC]"
23 Records selected
>LIST SALES WITH VALUE > "1000"
```

Select all the records in SALES file with an S.CODE value that starts with ABC. Then, using the list, report on the records in the SALES file which have a VALUE field greater than 1000.

EXAMPLE 2

```
SELECT SALES WITH S.CODE = "ABC]"
23 Records selected
>SAVE-LIST SALES.ABC
```

Select all the records in SALES file with an S.CODE value that starts with ABC. Then save the list as SALES.ABC.

SORT

Generates a sorted and formatted report of records and fields from a specified file

COMMAND SYNTAX

```
SORT file-specifier {record-list} {selection-criteria} {sort-
criteria} {USING file-specifier} {output-specification} {format-
specification} {(options}
```

Comments: Unless a different sort order is specified in the sort criteria, the presentation of the records will be in an ascending order based on the record key.
The data definition records (or the file definition records in the case of keys) determine whether to apply a left or right sort to the data.
If the field is left justified, it compares the data on a character-by-character basis from left to right, using ASCII values. **FOR EXAMPLE**:

```
01
100
21
A
ABC
BA
```

If the field is right justified and the data is numeric, it performs a numeric comparison and the values ordered by magnitude.
If the field is right justified and the data is alphanumeric, it collates the data into an alphanumeric sequence. For **EXAMPLE**:

```
A
01
123
ABCD
```

If a descending sequence is required, use the **BY-DSND** modifier in the sort criteria. Use the **BY-DSND** modifier with a data definition record to obtain a descending sequence of record keys, which points to field 0 (the key). See “Sort Criteria Clause” earlier for a full explanation of the sorting process.

EXAMPLE 1

```
SORT SALES
```

Sort all the records in the SALES file into key order and use the default data definition records (if found) to format the output.

EXAMPLE 2

```
SORT SALES WITH S.CODE = "ABC" "DEF" "GHI"
```

Select the records in the SALES file in which the S.CODE field contains the values ABC, DEF or GHI. Sort the records into key order.

EXAMPLE 3

```
GET-LIST SALES.Q4
SORT SALES GT "DEF" BY S.CODE
```

Get the implicit list called SALES.Q4 and, using the list, report on the records in the SALES file, which have a key greater than DEF. Sort does the report by S.CODE.

EXAMPLE 4

```
SORT SALES WITH S.CODE = "ABC]" OR "[DEF" BY-DSND S.KEY LPTR
```

Select the records in the SALES file in which the S.CODE field contains values which start with ABC or end with DEF. Sort the report in descending order of S.KEY (a data definition record which points to field 0 - the key) and output the report to the printer

EXAMPLE 5

```
SORT SALES BY S.CODE BREAK-ON S.CODE ""BL" P.CODE TOTAL VALUE GRAND-
TOTAL "Total" HEADING "Sales Code: "B" "DL" FOOTING "Page "CPP" LPTR
```

Sort the SALES file by S.CODE. Output the S.CODE, P.CODE and VALUE fields.
Control break on a change in **S.CODE** and suppress the LINE FEED before the break. Reserve the break value for use in the heading ("B"). Maintain a running total of the **VALUE** field and output it at each control break. Put the word "Total" on the grand-total line.
Set up a heading for each page which comprises the words "Sales Code: ", the sales code (from the break), a date and a **LINE FEED**. Set up a footing, which contains the text "Page", and a page number, centered on the line?
Produce the report on the currently assigned printer.

SORT-LABEL

Outputs data in a format suitable for producing labels

COMMAND SYNTAX

```
SORT-LABEL file-specifier {record-list} {selection-criteria} {sort-criteria} {USING file-specifier}{output-specification} {format-specification} {(options}
```

Prompts

At the prompt, supply formatting criteria as follows:

COL, ROW, SKIP, INDENT, SIZE, SPACE(,C):

COL	Number of columns required to list the data across the page.
ROW	Number of lines for each record; the output of each element of the output specification is on a separate line, if more elements exist in the output specification than there are rows specified it ignores the extra elements. If specifying more rows than elements, the output specification for these rows will be blank.
SKIP	Number of blank lines between each record.
INDENT	Number of spaces for left margin.
SIZE	Number of spaces required for the data under each column
SPACE	Number of horizontal spaces to skip between columns.
C	Optional. Suppresses null or missing data. If absent, null or missing values are output as blanks. If present, the C must be upper case and not in quotes.

Comments: The total number of columns specified must not exceed the page width, based on the calculation:

COLs * (SIZE + SPACE) + INDENT <= page width

ROW must be a minimum of one for each field, plus one for the record key (if not suppressed). If record keys are not suppressed, the first row of each label will contain the record key. If you specify a sort criteria clause, it sorts the records in key order.

If **INDENT** is not zero, at the prompt supply a series of **HEADER**s that will appear in the left margin for each field. If a heading is not required for a particular line, press **RETURN**.

Multivalued fields appear on separate lines.

If specified, **COL-HDR-SUPP** or **HDR-SUPP**, or the **C** or **H** options, the page number, date, and time will not be output and the generated report will be without page breaks.
See also the **LIST-LABEL** Command.

SREFORMAT

SREFORMAT is similar to the **SORT** Command in that it generates a formatted list of fields, but directs its output to another file or the magnetic tape rather than to the terminal or printer.

COMMAND SYNTAX

SREFORMAT file-specifier {record-list} {selection-criteria} {USING file-specifier} {output-specification} {format-specification} {(options}

Prompt: At the prompt supply the destination file:

File: Enter a file name, or the word "TAPE" for output to a magnetic tape.

Comments: Overwrites records that already exist in the destination file; when you reformat one file into another, each record selected becomes a record in the new file. It uses the first value specified in the output specification clause as the key for the new records. The remaining values in the output specification clause become fields in the new records.

When you reformat a file to tape, it concatenates the values specified in the output specification clause to form one tape record for each selected record. The record output is either truncated or padded at the end with nulls (X"00") to obtain a record the same length as specified when the tape was assigned by the T-ATT Command.

Unless you specify **HDR-SUPP** or **COL-HDR-SUPP**, or a **C** or **H** option, a tape label containing the file name, tape record length (in hexadecimal), it first writes the time, and date to the tape. If specifying a **HEADING** clause, this will form the data for the tape label.

Record keys are displayed as the records are written to tape unless the ID-SUPP modifier or the "I" option is specified.

Two EOF marks terminate the file on tape.

See the **REFORMAT** Command for Examples.

SSELECT

Generates an implicit list of record keys or specified fields, based on the selection criteria specified

COMMAND SYNTAX

```
SSELECT file-specifier {record-list} {selection-criteria} {sort-criteria} {output-criteria} {USING file-specifier} {(options}
```

SYNTAX ELEMENTS

Options are:

C{n}	Display running counters of the number of records selected and records processed. Unless modified by n, the counter increments after every 500 records processed or the total number of records if less than 500.
N	Specifies a number other than 500 by which to increment. For Example, C25 increments the counter after every 25 records processed.

Comments: Unless you specify a sort criteria clause it sorts the records in key order.

See also the **SELECT** Command.

If you specify an output-criteria clause, the generated list will comprise the data (field) values defined by the clause, rather than the selected record keys.

When the Command terminates, it displays the total number of entries in the generated list; the list is available to the next Command. This is indicated by the ">" prompt if you are in jSHELL.

If you use the **BY-EXP** or **BY-EXP-DSND** connectives on a multivalued field, the list will have the format:

record-key]multivalue#

where multivalue# is the position of the multivalue within the field specified by **BY-EXP** or **BY-EXP-DSND**. multivalue# can be accessed by a **READNEXT** Var,n statement in a jBASIC program.

EXAMPLE 1

```
SSELECT SALES WITH S.CODE = "ABC]"
23 Records selected
LIST SALES WITH VALUE > "1000"
```

Select all the records in SALES file with an S.CODE value that starts with ABC. Sort the list into key order. Then, using the list, report on the records in the SALES file which have a VALUE field greater than 1000.

EXAMPLE 2

```
SSELECT SALES WITH S.CODE = "ABC]" BY P.CODE
23 Records selected
```

```
>SAVE-LIST SALES.ABC
```

Select all the records in SALES file with an S.CODE value that starts with ABC. Sort the list into P.CODE order and then save the list as SALES.ABC.

Chapter 4 jBASIC Statements for Use with jQL

The creation of the following statements enable jBASIC programmers to deal directly with jQL statements, thereby eliminating the need to parse the output of commands such as EXECUTE.

NOTE: Properties are valid after the compile; this is the main reason for separating the compile and execute into two functions, after compiling, it is possible examine the properties and set properties before executing.

JQLCOMPILE

JQLCOMPILE compiles a jQL statement.

COMMAND SYNTAX

```
JQLCOMPILE(Statement, Command, Options, Messages)
```

SYNTAX ELEMENTS

Statement is the variable, which will receive the compiled statement (if it compiles); most other functions use this to execute and work on the result set etc.

Command is the actual jQL query that you want to compile (such as SELECT or something similar). Use RETRIEVE to obtain fetchable data records, as the verb rather than an existing jQL verb. This will ensure that the right options are set internally. In addition, use any word that is not a jQL reserved word as the verb and it will work in the same way as RETRIEVE: implement a PLOT command that passes the entire command line into JQLCOMPILE and the results will be the same as if the first word were replaced with RETRIEVE.

Option: To supply a select list to the JQLEXECUTE function specify JQLOPT_USE_SELECT; the compile builds a different execution plan if using select lists.

Messages: If the statement fails to compile, this dynamic array is in the STOP format, and therefore you can program and print STOP messages, which provides a very useful history of compilation for troubleshooting purposes. It returns -1 if a problem was found in the statement and zero if there was not.

JQLEXECUTE

JQLEXECUTE starts executing a compiled jQL statement.

COMMAND SYNTAX

```
JQLEXECUTE(Statement, SelectVar)
```

SYNTAX ELEMENTS

Statement is the valid result of a call to JQLCOMPILE(Statement, …).

SelectVar is a valid select list that used to limit the statement to a predefined set of items. For example:

```
: SELECT PROGRAMMERS WITH IQ_IN_PTS > 250
1 Item Selected
> LIST PROGRAMMERS NAME

PROGRAMMERS...     NAME
0123               COOPER, F B
```

This function returns -1 in the event of a problem, such as the statement variable not being correct. It will cause the statement to run against the database and produce a result set for use with JQLFETCH()

JQLFETCH

JQLFETCH fetches the next result in a compiled jQL statement.

COMMAND SYNTAX

```
JQLFETCH(Statement, ControlVar, DataVar)
```

SYNTAX ELEMENTS

Statement is the result of a valid call to JQLCOMPILE(), followed by a valid call to JQLEXECUTE().

ControlVar will receive the 'control break' elements of any query. For example, if there are BREAK values in the statement, and you want the totals, they will be described here.

The format of ControlVar is:

```
Attr 1      Level:
0 means detail line
1 - 255     for the control breaks, the same as the A correlative
NB.
Attr2       Item ID
Attr 3      Break control
Value is 1 if a blank line should be output first.
Attr 4      Pre-break value for 'B' option in header
Attr 5      Post-break value for 'B' option in header
```

DataVar will receive the actual data sent to the screen on a LIST statement for instance. The format is one attribute per column.

Applies Attribute 7 Conversions (or attribute 3 in Prime-style DICTS) to the data

If setting the property STMT_PROPERTY_FORMAT then it also formats each attribute according to the width and justification of the attribute definition and any override caused by the use of FMT, of DISPLY.LIKE on the command line –

NOTE: that column headers may also affect the formatting for that column.

This function is called multiple times until there is no more output.

JQLGETPROPERTY

Gets the property of a compiled jQL statement

COMMAND SYNTAX

```
JQLGETPROPERTY(PropertyValue, Statement, Column, PropertyName)
```

SYNTAX ELEMENTS

Option	Description
PropertyValue	Receives the requested property value from the system or “” if the property is not set
Statement	The result of a valid JQLCOMPILE(Statement)
Column	Specifies that you want the value of the property for a specific column (otherwise 0 for the whole statement).
PropertyName	These are EQUATED values defined by INCLUDE’ing the file JQLINTERFACE.h. This function returns -1 if there is a problem with the parameters or the programmer. The use of these properties is to answer questions such as “Was LPTR mode asked for,” and “How many columns are there?”

NOTE: Properties are valid after the compile; this is the main reason for separating the compile and execute into two functions. After compiling, it is possible examine the properties and set properties before executing.

JQLPUTPROPERTY

Sets a property in a compiled jQL statement

COMMAND SYNTAX

```
JQLPUTPROPERTY(PropertyValue, Statement, Column, PropertyName)
```

SYNTAX ELEMENTS

PropertyValue is the value you want to which to set the specified property, such as 1 or "BLAH"

Statement is the result of a valid JQLCOMPILE() function.

Note: Some properties may require JQLEXECUTE() first.

Column Holds 0 for a general property of the statement, or a column number if it is something that can be set for a specific column.

PropertyName – These are EQUATED values defined by INCLUDE'ing the file JQLINTERFACE.h. There are lots of these and someone is going to have to document each one.

This function returns -1 if a problem was found in the statement and 0 if there was not.

NOTE: Properties are valid after the compile; this is the main reason for separating the compile and execute into two functions. After compiling, it is possible examine the properties and set properties before executing.

Chapter 5 jQL Conversion Codes

Conversion Processing

A	Algebraic functions.
B	Subroutine call.
C	Concatenation.
D	Internal and external dates.
D1 and D2	Associates controlling and dependent fields.
F	Mathematical functions.
G	Group extract.
L	Length.
MC	Mask character.
MD	Mask decimal.
MK	Mask metric.
ML and MR	Mask with justification.
MP	Mask packed decimal.
MS	Mask Sequence.
MT	Mask time.
P	Pattern match.
R	Range check.
S	Substitution.
T	Text extraction.
TFILE	File translation.
U	User exit.

W	Timestamps.
JBCUserConversions	How to create user-defined conversion codes

jQL Dictionary Conversions and Correlatives

For dates and times, simple date format functions have been applied to use the configured locale to support the standard conversions D and MTS. Formatting numbers via MR/ML/MD, use locale for Thousands, Decimal Point and Currency notation.

TimeStamp “W{Dx}{Tx}”

In addition, to provide for timestamp functionality included is a suite of conversions including A, F and I types. This is to generate a timestamp, displayed for date and/or time in short, long, and full formats. These conversions also support non-Gregorian locales. The meaning of the components of the conversion is as follows:

W	Is a new conversion code so not to clash with existing conversions.
D	Date
T -	Time
x	Format option: S = Short, M = Medium, L = Long, F = Full

“WDS” or “WTS” SHORT is completely numeric.**12/13/52 or 3:30pm**
“WDM” MEDIUM is longer. **Jan 12, 1952**
“WDL” or “WTL” LONG is longer.
January 12, 1952 or 3:30:32pm
“WDF” or “WTF” FULL is specified completely.

Data Conversion

When executing programs in international mode, it processes all variable contents as UTF-8 encoded sequences. As such all data must be held as UTF-8 encoded byte sequences. This means that data imported into an account configured to operate in international mode must be converted from the data in the current code page to UTF-8. Normally if ALL the data are eight bit bytes in the range 0x00-0x7f (ASCII) then no conversion is necessary as these values are effectively already UTF-8 encoded. However values outside of the 0x00-0x7f range must be converted into UTF-8 proper such that there can be no ambiguity between character set code page values.

For instance, the character represented by the hex value 0xE0 in the Latin2 code page, (ISO-8859-2), is described as “LATIN SMALL LETTER R WITH ACUTE”. However the same hex value in the Latin1 code page, (ISO-8859-1), is used to represent the character “LATIN SMALL LETTER A WITH GRAVE”.
To avoid this clash of code pages the Unicode specification provides unique hex value representations for both of these characters within the specifications 32-bit value sequence.

EXAMPLE

Unicode value 0x00E0 used to represent LATIN SMALL LETTER A WITH GRAVE
Unicode value 0x0155 used to represent LATIN SMALL LETTER R WITH ACUTE
NOTE: that UTF-8 is an encoding of 32 bit Unicode values, which also has especially properties (as described earlier), which can be used effectively with Unix and Windows platforms.
Another good reason for complete conversion from the original code page to UTF-8 is that doing so also removes the requirement for conversions when reading/writing to files, as this would add massive and unnecessary overhead to ALL application processing, whereas the conversion from original code page to UTF-8 is a one off cost.

A Conversion

"A" codes provide many powerful features, which include arithmetic, relational, logical, and concatenation operators, the ability to reference fields by name or FMC, the capability to use other data definition records as functions that return a value, and the ability to modify report data by using format codes.
The A code also allows you to handle the data recursively, or “nest” one A code expression inside another.

SYNTAX SUMMARY

The A code function uses an algebraic format. There are two forms of the A code:

- A uses only the integer parts of stored numbers unless a scaling factor is included.
- AE handles extended numbers. Uses both integer and fractional parts of stored numbers.

COMMAND SYNTAX

```
A{n}{;expression}
AE;expression
```

SYNTAX ELEMENTS

n	is a number from 1 to 6 that specifies the required scaling factor.
expression	Comprise operands, operators, conditional statements, and special functions.

Comments: The A code replaces and enhances the functionality of the F code

A;expression	evaluates the expression.
An	converts to a scaled integer
An;expression	converts to a scaled integer.
AE;expression	evaluates the expression.

A: Expression Format

Performs the functions specified in expression on values stored without an embedded decimal point.

An Format: Embedded Decimals

The "An" format converts a value stored with an embedded decimal point to a scaled integer. The stored value's explicit or implied decimal point is moved n digits to the right with zeros added if necessary. Returns only the integer portion
Field 2 of the data definition record must contain the FMC of the field that contains the data to be processed.

An;expression Format

The "An;expression" format performs the functions specified in expression on values stored with an embedded decimal point. It then converts the resulting value to a scaled integer.

AE;expression Format

The AE format uses both the integer and fractional parts of stored numbers. Use format codes to scale do scaling of output..

Examples of Numeric Results

Data Record		A Code		
Field 1	Field 2	A;1 + 2	A3;1 + 2	AE;1 + 2
4	012	16	16000	16
-77	-22	-99	-99000	-99
0.12	22.09	22	22210	22.21
-1.234	-12.34	-13	-13574	-13.574
-1.234	123.45	122	122216	122.216

Does not allow Input Conversion

Format Codes

You can format the result of any "A" code operation by following the expression with a value mark, and then the required format code:
An;expression]format
Format codes can also be included within the expression. For more information, see Format codes.

Summary of Operands

Operands, which you can use in "A" code expressions include:

- FMCs (field numbers),
- field names
- literals,
- operands that return system parameters,
- special functions

You can format any operand by following it with one or more format codes enclosed in parentheses, and separated by value marks, (ctrl]):
operand(format-code{]format-code}...)

For more information, see Format Codes.

Field Number (FMC) Operand

The field number operand returns the content of a specified field in the data record:
field-number{R{R}}

The first **R** specifies that any non-existent multivalues should use the previous non-null multivalue. When the second **R** is specified, any non-existent subvalues should use the previous non-null subvalue.

Field Name Operand

The field name operand returns the content of a specified field in the data record:

N(field-name){R{R}}

Literal Operand

The literal operand supplies a literal text string or numeric value:

“literal”

System Parameter Operands

Several A code operands return the value of system parameters. They are:

D	Returns the system date in internal format.
LPV	Returns the previous value transformed by a format code.
NA	Returns the number of fields in the record.
NB	Returns the current break level counter. 1 is the lowest break level, 255 is the GRAND TOTAL line.
ND	Returns the number of records (detail lines) since the last control break.
NI	Returns the record counter.
NL	Returns the record length in bytes
NS	Returns the subvalue counter
NU	Returns the date of last update
NV	Returns the value counter
T	Returns the system time in internal format.
V	Returns the previous value transformed by a format code

Special Operands

Some operands allow you to use special functions. They are:

I(expression)	Returns the integer part of expression.
R(exp1, exp2)	Returns the remainder of exp1 divided by exp2.
S(expression)	Returns the sum of all values generated by expression.
string[start-char-no, len]	Returns the substring starting at character start-char-no for length len.

Summary of Operators

Operators used in A code expressions include arithmetic, relational and logical operators, the concatenation operator, and the IF statement.

Arithmetic Operators

Arithmetic operators are:

+	Sum of operands
-	Difference of operands
*	product of operand
/	Quotient (an integer value) of operands

Relational Operators

Relational operators specify relational operations so that any two expressions can treated as operands and evaluated as returning true (1) or false (0). Relational operators are:

= or EQ	Equal to

< or LT	Less than
> or GT	Greater than
<= or LE	Less than or equal to
>= or GE	greater than or equal to
# or NE	Not equal

Logical Operators

The logical operators test two expressions for true or false and return a value of true or false. Logical operators are:

AND	Returns true if both expressions are true.
OR	Returns true if any expressions is true.

Concatenation Operator

The concatenation operator is a colon (:)

IF STATEMENT

The IF operator gives the A code its conditional capabilities. An IF statement looks like this: IF expression THEN statement ELSE statement
Field Number (FMC) Operand specifies a field, which contains the value for use.

COMMAND SYNTAX

```
field-number{R{R}}
```

SYNTAX ELEMENTS

field-number	the number of the field (FMC), which contains the required value.
R	specifies that the value obtained from this field be applied for each multivalue not present in a corresponding part of the calculation.
RR	Specifies that the value obtained from this field be applied for each subvalue not present in a corresponding part of the calculation.

Comments: The following field numbers have special meanings:

0	Record key
9998	Sequential record count
9999	Record size in bytes

EXAMPLE 1

A;2

Returns the value stored in field 2 of the record.

EXAMPLE 2

A;9999

Returns the size of the record in bytes

EXAMPLE 3

A;2 + 3R

For each multivalue in field 2, the system also obtains the (first) value in field 3 and adds it. If field 2 contains 1]7 and field 3 contains 5 the result would be two values of 6 and 12 respectively. Where three does not have a corresponding multivalue, will use the last non-null multivalue in three

EXAMPLE 4

A;2 + 3RR

For each subvalue in field 2, the system also obtains the corresponding subvalue in field 3 and adds it. If field 2 contains 1\2\3]7 and field 3 contains 5\4 the result would be four values of 6, 6, 7, 12 and 4 respectively.

N (Field Name) Operand

References another field defined by a name in the same dictionary or found in one of the default dictionaries.

COMMAND SYNTAX

```
N(field-name){R{R}}
```

SYNTAX ELEMENTS

field-name	is the name of another field defined in the same dictionary or found in the list of default dictionaries
R	Specifies that the value obtained from this field be applied for each multivalue not present in a corresponding part of the calculation.
RR	Specifies that the value obtained from this field be applied for each subvalue not present in a corresponding part of the calculation.

Comments: If the data definition record of the specified field contains field eight pre-process conversion codes, it applies these before it returns the value(s).
Any pre-process conversion codes in the specified field-name including any further N(field-name) constructs are processed as part of the conversion code.
N(field-name) you can nest constructs up to 30 levels. The number of levels is restricted to prevent infinite processing loops. For Example:

TEST1

```
008 A;N(TEST2)
```

TEST2

```
008 A;N(TEST1)
```

EXAMPLE 1

```
A;N(S.CODE)
```

Returns the value stored in the field defined by S.CODE.

EXAMPLE 2

```
A;N(A.VALUE) + N(B.VALUE)R
```

For each multivalue in the field defined by A.VALUE, the system also obtains the corresponding value in B.VALUE and adds it. If A.VALUE returns 1]7 and B.VALUE returns 5, the result would be two values of 6 and 12 respectively.

EXAMPLE 3

```
A;N(A.VALUE) + N(B.VALUE)RR
```

For each subvalue in the field defined by A.VALUE, the system also obtains the corresponding value in B.VALUE and adds it. If A.VALUE returns 1\2\3]7 and B.VALUE returns 5 the result would be four values of 6, 7, 8 and 12 respectively.

Literal Operand

Specifies a literal string or numeric constant enclosed in double quotes

COMMAND SYNTAX

```
"literal"
```

SYNTAX ELEMENTS

literal is a text string or a numeric constant.

NOTES

Assumes a number not enclosed in double quotes to be a field number (FMC).

EXAMPLE 1

```
A;N(S.CODE) + "100"
```

Adds 100 to each value (subvalue) in the field defined by S.CODE

EXAMPLE 2

```
A;N(S.CODE):"SUFFIX"
```

Concatenates the string "SUFFIX" to each value (subvalue) returned by S.CODE

Special Operands

Integer Function: I(expression) returns the integer portion of an expression.

EXAMPLE

```
AE;I(N(COST) * N(QTY))
```

Returns the integer portion of the result of the calculation

Remainder Function

The Remainder Function R(exp1, exp2) takes two expressions as operands and returns the remainder when dividing the first expression by the second.

Example: A;R(N(HOURS) / “24”) - Returns the remainders when 24 divide HOURS.

Summation Function: S(expression) evaluates an expression and then adds together all the values.

EXAMPLE

```
A;S(N(HOURS) * N(RATE)R)
```

Multiplies each value in the HOURS field by the value of RATE; the multivalued list of results is then totalled.

Substring Function

The substring function [start-char-no, len] extracts the specified number of characters from a string, starting at a specified character.

SYNTAX ELEMENTS

Start-char no	An expression that evaluates to the position of the first character of the substring.
Len	An expression that evaluates to the number of characters required in the substring. Use - len (minus prefix) to specify the end of the substring. For Example, [1, -2] will return all but the last character and [-3, 3] will return the last three characters.

EXAMPLE 1

```
A;N(S.CODE)["2", "3"]
```

Extracts a sub-string from the S.CODE field, starting at character position 2 and continuing for 3 characters

EXAMPLE 2

```
A;N(S.CODE)[2, N(SUB.CODE.LEN)]
```

Extracts a sub-string from the S.CODE field, starting at the character position defined by field 2 and continuing for the number of characters defined by SUB.CODE.LEN

Format Codes: Specifies a format code to be applied to the result of the A code or an operand.

COMMAND SYNTAX

```
a-code{]format-code...}
a-operand(format-code{]format-code}...)
```

SYNTAX ELEMENTS

A code	A complete A Code expression.
A operand	One of the A Code operands.
format code	is one of the codes described later G(roup), D(ate) or M(ask).
]	represents a must use value mark to separate each format code.

Comments: You can format the result of the complete "A" code operation by following the expression with a value mark and then the required format code(s). (This is a standard feature of the data definition records.)

Format codes can also be included within "A" code expressions; enclosed in parentheses, using a value mark for separation if using more than one format code. All format codes will convert values from an internal format to an output format.

EXAMPLE 1

```
A;N(COST)(MD2]G0.1) * ...
```

Shows two format codes applied within an expression. Obtains the COST value and applies an MD2 format code. Then applies a group extract to acquire the integer portion of the formatted value. You can now use the integer portion in the rest of the calculation. Could also have been achieved like this:

```
A;I(N(COST)(MD2)) * ...
```

EXAMPLE 2

```
A;N(COST) * N(QTY)]MD2
```

Shows the MD2 format code applied outside the A code expression. Multiplies COST by QTY and the result formatted by the MD2 format code.

Operators

Operators used in A code expressions include arithmetic, relational and logical operators, the concatenation operator, and the IF statement.

ARITHMETIC OPERATORS

Arithmetic operators are:

+	Sum of operands
-	Difference of operands
*	Product of operands
/	Quotient (an integer value) of operands

Relational Operators

Relational operators specify relational operations so that any two expressions can treated as operands and evaluated as returning true (1)
or false (0). Relational operators are:

= or EQ	Equal to
< or LT	Less than
> or GT	Greater than
<= or LE	Less than or equal to
>= or GE	greater than or equal to
# or NE	Not equal

Logical Operators

The logical operators test two expressions for true (1) or false (0) and return a value of true or false. Logical operators are:

AND|Returns	true if both expressions are true.
OR|Returns	True if any expressions is true.

The words **AND** and **OR** must be followed by at least one space. The AND operator takes precedence over the OR unless you specify a different order by means of parentheses. OR is the default operation.

Concatenation Operator

Use a colon (:) to concatenate the results of two expressions.
For Example: the following expression concatenates the character “Z” with the result of adding together fields 2 and 3:
A;”Z”:2 + 3
IF Statement: gives the A code conditional capabilities

COMMAND SYNTAX

```
IF expression THEN statement ELSE statement
```

SYNTAX ELEMENTS

expression	must evaluate to true or false. If true, executes the THEN statement. If false, executes the ELSE statement.
statement	is a string or numeric value.

Comments: Each IF statement must have a THEN clause and a corresponding ELSE clause. You can nest statements but the result of the statement must evaluate to a single value. The words IF, THEN and ELSE must be followed by at least one space.

EXAMPLE 1

```
A;IF N(QTY) < 100 THEN N(QTY) ELSE ERROR!
```

Tests the QTY value to see if it is less than 100. If it is, output the QTY field. Otherwise, output the text “ERROR!”.

EXAMPLE 2

```
A;IF N(QTY) < 100 AND N(COST) < 1000 THEN N(QTY) ELSE ERROR!
Same as Example 1 except that QTY will only be output if it is less
than 100 and the cost value is less than 1000.
```

EXAMPLE 3

```
A;IF 1 THEN IF 2 THEN 3 ELSE 4 ELSE 5
```

If field 1 is zero or null, follow else and use field 5. Else test field 2; if field 2 is zero or null, follow else and use field 4. Else, use field 3. Use Field 3 only if both fields 1 and 2 contain a value.

B Conversion

Provides interface for jBASIC subroutines or C functions to manipulate data during jQL processing.

Synonymous with CALL code

See Calling Subroutines from Dictionary Items for more details.

C Conversion

Concatenates fields, literals, and the results of a previous operation

COMMAND SYNTAX

```
C{;}n{xn}...
```

SYNTAX ELEMENTS

;	Optional in that it has no function other than to provide compatibility.
x	The character for insertion between the concatenated elements. If you specify a semicolon (;), no separator will be used. Any non-numeric character except system delimiters (value, subvalue, field, start buffer, and segment marks) is valid.

n can be any one of the following:

field number (FMC)

a literal enclosed in single quotes, double quotes, or backslashes

an asterisk (*) to specify the last generated value of a previous operation

Comments: See the descriptions of the function codes (A, F, FS and their variants) for other concatenation methods.

Input Conversion: does not invert; applies the concatenation to the input data.

EXAMPLE 1

```
C1;2
```

Concatenates the contents of field 1 with field 2, with no intervening separator character

EXAMPLE 2

```
C1*2
```

Concatenates the contents of field 1 with an asterisk (*) and then the content of field 2

EXAMPLE 3

```
C1*"ABC" 2/3
```

Concatenates the contents of field 1 with an asterisk (*), the string ABC, a space, field 2 a forward slash (/) and then field 3.

D Conversion

Converts dates between internal and external format.

COMMAND SYNTAX

```
D{p}{n}{s}
```

SYNTAX ELEMENTS

P	The special processing operator and can be any one of the following:
D	Returns only the day of the month as a numeric value.
I	Returns only dates stored in the external format in internal format. You can use this in field 7 or 8.
J	Returns the Julian day (1 - 365, or 1 - 366 for a leap year).
M	Returns the number of the month (1 - 12).
MA	Returns the name of the month in uppercase letters.
Q	Returns the number of the quarter (1 - 4)
W	Returns the day of the week as a numeric value (Monday is 1).
WA	Returns the day of the week in uppercase letters (MONDAY - SUNDAY).
Y	Returns the year (up to four digits).
n	is a number from 0 to 4 that specifies the how many digits to use for the year field. If omitted, the year will have four digits; suppresses the year if n is 0.
s	used as a non-numeric character as a separator between month, date, and year. Must not be one of the special processing operators.

Comments: Dates are stored internally as integers, which represent the number of days (plus or minus) from the base date of December 31, 1967. For Example:

Date	Stored Value

22 September 1967	-100
30 December 1967	-1
31 December 1967	0
01 January 1968	1
09 April 1968	100
26 September 1967	1000
14 January 1995	9876
29 February 2000	11748

If you do not specify a special processing operator (see later) or an output separator, the default output format is two-digit day, a space, a three-character month, a space, and a four-digit year. If you specify just an output separator, the date format defaults either to the US numeric format “mm/dd/yyyy” or to the international numeric format “dd/mm/yyyy” (where / is the separator). You can change the numeric format for the duration of a logon session with the DATE-FORMAT Command.

Pre-processor Conversion

Field 8 codes are valid but, generally, it is easier to specify the D code in field 7 for input conversion. Dates in output format are difficult to use in selection processing.
If you are going to use selection processing and you want to use a code which reduces the date to one of its parts, such as DD (day of month), the D code must be specified in field 8.
Field 7 input and output conversions are both valid.
Generally, for selection processing, you should specify D codes in field 7 except when you use a formatting code, such as DM, that reduces the date to one of its parts. If you specify no year in the sentence, the system assumes the current year on input conversion. If specifying only the last two digits of the year the system assumes the following:

00-29	2000-2029
30-99	1930-1999

EXAMPLES

D Code	Internal Value	Value Returned
D	9904	11 FEB 1995
D2/	9904	11/02/95
D-	9904	11-02-1995
D0	9904	11 FEB
DD	9904	11
DI	11 FEB 1995	9904
DJ	9904	41
DM	9904	2
DMA	9904	FEB
DQ	9904	1
DW	9904	6
DWA	9904	SATURDAY
DY	9904	1995
DY2	9904	95

D1 D2 Conversion

Associates controlling and dependent fields

COMMAND SYNTAX

```
D1;fmcd{;fmcd}...
D2;fmcc
```

SYNTAX ELEMENTS

fmcd is the field number (FMC) of an associated dependent field.

fmcc is the field number (FMC) of the associated controlling field.

Comments: You can logically group multivalued fields in a record by using a controlling multivalued field and associating other fields with it. For example, you could group the component parts of an assembly on an invoice.
The D1 code in field 8 defines the controlling field and nominates the associated dependent fields. Each dependent field will have a D2 code in field 8.
Important: The D1 and D2 codes must be in field 8 of the data definition record and be the first code specified; other codes can follow (separated by a value mark), but it must be the first code.
Outputs the values in the dependent associative fields in order as specified in field 8 of the controlling field the specified order in the dependent fields in the output specification clause is irrelevant.

EXAMPLE:

```
LIST INVOICE "ABC123" PARTS QTY PRICE
```

The records in data file INVOICE have three associated, multivalued fields, named PARTS, QTY and PRICE, and numbered seven, two and five respectively.
PARTS is the controlling field because, for each multivalue in this field there will a corresponding value in the other fields, and also because PARTS should appear first on the report. The data definition record for PARTS will have D1;2;5 in field 8.
The data definition records for QTY and PRICE will both have D2;7 in field eight.
The report generated by the Command will look something like this:

```
INVOICE PART QTY PRICE
ABC123 AAA 1 10.00
```

```
BBB 11 4.00
CCC 2 3.30
```

F Conversion

F codes provide many facilities for arithmetic, relational, logical, and concatenation operations. The expression of all operations is in Reverse Polish notation and involves the use of a “stack” to manipulate the data.

SYNTAX SUMMARY

There are three forms of the F code:

F Uses only the integer parts of stored numbers unless a scaling factor is included. If the JBCEMULATE environment variable is set to “ROS” the operands for “-”, “/” and concatenate are used in the reverse order.

FS Uses only the integer parts of stored numbers (use SMA standard stack operations for all emulations)

FE Uses both the integer and fraction parts of stored numbers.

COMMAND SYNTAX

```
F{n};elem{;elem}...
FS;elem{;elem}...
FE;elem{;elem}
```

SYNTAX ELEMENTS

n	A number from 1 to 9 used to convert a stored value to a scaled integer. The stored value explicit or implied decimal point is moved n digits to the right with zeros added if necessary. Returns only the integer portion of this operation.
elem	Any valid operator

Comments: F codes use the Reverse Polish notation system. Reverse Polish is a postfix notation system where the operator follows the operands. The expression for adding two Elements is “a b + “. (The usual algebraic system is an infix notation where the operator is placed between the operands, for Example, “a + b”).

The F code has operators to push operands on the stack. Other operators perform arithmetic, relational, and logical operations on stack Elements. There are also concatenation and string operators.

Operands pushed on the stack may be constants, field values, system parameters (such as date and time), or counters (such as record counters).

The Stack

F codes work with a pushdown stack.

NOTE: All possible F correlative operators push values onto the stack, perform arithmetic and other operations on the stack entries, and pop values off the stack.
The term “push” is used to indicate the placing of an entry (a value) onto the top of the stack so that existing entries are pushed down one level. “Pop” means to remove an entry from the top of the stack so that existing entries pop up by one level. Arithmetic functions typically begin by pushing two or more entries onto the stack. Each operation then pops the top two entries, and pushes the result back onto the top of the stack. After any operation is complete, the result will always be contained in entry 1.

Order of Operation

F code operations are typically expressed as “F;stack2;stack1;operation” and evaluated under most emulation, as “stack2 operation stack1”.
If JBCEMULATE is set to “ROS”, this example is evaluated as “stack1 operation stack2”, effectively reversing the order of operations.

NOTE: that the FE and FS codes are evaluated in the same way for all emulations.
No Input conversion allowed.

EXAMPLE 1

```
F;C3;C5;-
```

Push a value of three onto the stack. Push a value of five onto the stack.
Take entry 1 from entry 2 (3 - 5) and push the result (-2) back onto the stack as entry 1. ROS emulations will subtract 3 from 5 and return a result of two.

EXAMPLE 2

```
FS;C3;C5;-
```

Push a value of three onto the stack. Push a value of five onto the stack. Take entry 2 from entry 1 (3 - 5) and push the result (-2) back onto the stack. This works in the same way for all emulations.

EXAMPLE 3

```
F;C2;C11;C3;-;/
```

Push a value of two onto the stack. Push a value of 11 onto the stack. Push a value of three onto the stack. Subtract entry 1 from entry 2 (11 - 3) and push the result (8) back onto the stack. Now divide entry 2 by entry 1 (2 divided by 8) and push the result (0) back onto the stack.

Under ROS emulation, this would evaluate as 3 - 11 = -8, followed by -8 / 2 = -4.

Push Operators

A push operator always pushes a single entry onto the stack. Existing entries are moved one position down. Push operators are: “literal” Literal. Any text string enclosed in double or single quotes. field-number{R{R}}{(format-code)}

Field number	The value of the field from the current record.
R	Specifies that the last non-null value obtained from this field be applied for each multivalue that does not exist in a corresponding part of the calculation.
RR	Specifies that the last non-null value obtained from this field be applied for each subvalue that does not exist in a corresponding part of the calculation.
(format code)	One or more format codes (separated by value marks) enclosed in parentheses and applied to the value before it is pushed onto the stack.
Cn	Constant - where n is a constant (text or number) of any length up to the next semicolon or system delimiter.
D	Current system date.
NA	Number of fields in the record.
NB	Current break level number: -1 = during SORT or SELECT processing 0 = detail line 1 = lowest break level 255 = GRAND-TOTAL line
ND	Number of records since the last BREAK on a BREAK data line. Equal to the record counter on a GRAND-TOTAL line. Used to compute averages.
NI	Record counter. The ordinal position of the current record in the report.
NL	Length of the record, in bytes. Includes all field marks but not the key.
NS	Subvalue counter. The ordinal position of the current subvalue within the field.
NV	Value Counter. The ordinal position of the current multivalue within the field.
P	Alternatively, it pushes duplicate of entry 1 onto the stack.

T	System time in internal format.
V or LPV	Previous Value. Use the value from the previous format code.

Arithmetic F Code Operators

The arithmetic F code operators work on just the top stack entry or the top two stack entries. They are:

+	Add the top two stack entries together and push result into entry 1.
-	Subtract stack entries and push result into entry 1: F - subtract entry 1 from entry 2 FS, FE - subtract entry 1 from entry 2 F - subtract entry 2 from entry 1 (ROS emulation)
*{n}	Multiply the top two stack entries and push result into entry 1. If n is specified, the result is divided by 10 raised to the power of n.
/	Divide stack entries and push quotient into entry 1: F - divide entry 2 by entry 1 FS, FE - divide entry 2 by entry 1 F - divide entry 1 by entry 2 (ROS emulation)
R	Compute remainder from the top two stack entries and push result into entry 1: F - remainder of entry 2 / entry 1 FS, FE - remainder of entry 2 / entry 1 F - remainder of entry 1 / entry 2
I	Return the integer part of entry 1 to the top of the stack.
S	Replace the multivalued entry 1 with the sum of the multivalues and subvalues.

Miscellaneous Operators

Miscellaneous operators control formatting, exchanging stack entries, popping the top entry, concatenation, and string extraction. They are:

_	Exchange the top two entries.
^	Pop last entry from the stack and discard. Pushes all other entries up.
(format-code)	Perform the specified format code on last entry and replace last entry with the result .
:	Concatenate stack entries: F - Concatenates Entry 1 to the end of Entry 2 FS, FE - Concatenates Entry 1 to the end of Entry 2 F - Concatenates Entry 2 to the end of Entry 1(ROS emulation)
[ ]	Extract a substring from stack entry 3. The starting column is specified in stack entry 2 and the number of characters is specified in entry 1

Relational Operators

Relational operators compare stack entries and push the result onto stack entry 1; is either 1 (true) or 0 (false). Relational operators are:

=	equal to
<	less than: F entry 2 < entry 1 FS,FE entry2 < entry 1 F entry 1 < entry 2 (ROS emulation)
>	Greater than: Fentry 2 > entry 1 FS,FE entry2 > entry 1 F entry 1 > entry 2 (ROS emulation)
[	Less than or equal to: F entry 2 [entry 1 FS,FE entry2 [entry 1 F entry 1 [entry 2 (ROS emulation)

]	Great than or equal to: F entry 2 [entry 1 FS,FE entry 2 [entry 1 F entry 1 [entry 2 (ROS emulation)
#	Not equal.

Logical Operators

Logical operators include a logical AND test and a logical inclusive-OR test.

Logical operators are:

& AND stack entries 1 and 2.	If both entries contain non-zero, pushes a 1 onto stack entry 1, otherwise, pushes a 0.
! OR stack entries 1 and 2.	If either of the entries contains non-zero, it pushes a 1 onto stack entry 1; otherwise, pushes a 0.

Multivalues

A powerful feature of F and FS code operations is their ability to manipulate multivalues. Individual multivalues can be processed, one by one, or you can use the R (or RR) modifier after a field number, to repeat a value and thus combine it with each of a series of multivalues. Field operands may be valued and subvalued. When performing mathematical operations on two multivalued lists (vectors), the result is also multivalued. The result has many values as the longer of the two lists. Zeros are substituted for the null values in the shorter list if the R option is not specified.

Repeat Operators

To repeat a value for combination with multivalues, follow the field number with the R operator. To repeat a value for combination with multiple subvalues, follow the FMC with the RR operator.

Format Codes

There are three ways to use format codes:

1. One transforms the result of the F code
2. Another transforms the content of a field before it is pushed on the stack
3. The third transforms the top entry on the stack

COMMAND SYNTAX

```
f-code{]format-code...}
field-number(format-code{]format-code}...)
(format-code{]format-code}...)
```

SYNTAX ELEMENTS

F code	A complete F Code expression.
Field number	The field number in the record from which to retrieve the data.
format code	Any valid format codes.
]	Represents a must use value mark (ctrl]) to separate each format code.

Comments: To process a field before it is pushed on the stack, follow the FMC with the format codes enclosed in parentheses. To process the top entry on the stack, specify the format codes within parentheses as an operation by itself. To specify more than one format code in one operation, separate the codes with the value mark, (ctrl]). All format codes will convert values from an internal format to an output format.

EXAMPLE

```
F;2(MD2]G0.1);100;-
```

Obtain the value of field 2. Apply an MD2 format code. Then apply a group extract to acquire the integer portion of the formatted value, and push the result onto the stack. Subtract 100 from the field 2 formatted, group extracted value. Return this value. Note that under ROS emulation, the value returned would be the result of subtracting the integer value from the group extract, from 100. In other words:

```
100 - OCONV(OCONV(Field2, "MD2"), "G0.1" ).
```

G Conversion

G codes extract one or more contiguous strings (separated by a specified character), from a field value.

COMMAND SYNTAX

```
G{m}xn
```

SYNTAX ELEMENTS

m	the number of strings to skip. If omitted or zero, extraction begins with the first character.
x	the separation character.
n	the number of strings to be extracted.

Comments: The field value can consist of any number of strings, each separated by the specified character. The separator can be any non-numeric character, except a system delimiter.
If m is zero or null and the separator x is not found, the whole field will be returned. If m is not zero or null and the separator x is not found, null will be returned.
Input Conversion: does not invert. It simply applies the group extraction to the input data.

EXAMPLE 1

```
G0.1
```

If the field contains “123.45”, 123 will be returned. You could also use “G.1” to achieve the same effect.

EXAMPLE 2

```
G2/1
```

If the field contains “ABC/DEF/GHI”, returns GHI.

EXAMPLE 3

```
G0,3
```

If the field contains “ABC,DEF,GHI,JKL”, returns ABC,DEF,GHI. Note that the field separators are included in the returned string.

L Conversion

L codes return the length of a value, or the value if it is within specified criteria.

COMMAND SYNTAX

```
L{{min,}max}
```

SYNTAX ELEMENTS

min	Specifies that the process is to return an element if its length is greater than or equal to the number min.
max	Specifies that the process is to return an element if its length is less than or equal to the number max.

Comments: The L code by itself returns the length of an element. When used with max, or min and max, the L code returns the element if it is within the length specified by min and/or max.

EXAMPLE 1

L - Assuming a value of ABCDEF, returns the value 6.

EXAMPLE 2

```
L4
```

If JBCEMULATE is set to ROS, L4 is translated as return the value if its length is less than or equal to 4 - the equivalent of L0,4. Assuming a value of ABCDEF, L4 will return null - the value is longer than 4 characters.

If JBCEMULATE is not set to ROS, L4 is translated as return the value if its length is exactly equal to 4 - the equivalent of L4,4. Assuming a value of ABCDEF, L4 will return null - the value is longer than 4 characters.

EXAMPLE 3

```
L4,7
```

L4,7 is translated as return the value if its length is greater than or equal to 4 and less than or equal to 7. Assuming a value of ABCDEF, L4,7 will return ABCDEF.

MC Conversion

MC codes include facilities for:

1. Changing characters to upper or lower case

2. Extracting a class of characters
3. Replacing characters
4. Converting ASCII character codes to their hexadecimal or binary representations and vice versa
5. Converting a hexadecimal values to their decimal or binary equivalents and vice versa
6. Converting a decimal value to its equivalent in Roman numerals and vice versa

One source of confusion when using MC codes is that input conversion does not always invert the code. If most MC codes are used in field 7 of the data definition record, applies the code in its original (un-inverted) form to the input data Therefore, you should always try to place MC codes into field 8 of the data definition record. The exceptions to this, is where input conversion is effective, are clearly indicated in the following sections.

SUMMARY:

MC codes are:

MCA	Extract only alphabetic characters
MC/A	Extract only non-alphabetic characters.
MCAB{S}	Convert ASCII character codes to binary representation. Use S to suppress spaces.
MCAX or MX	Convert ASCII character codes to hexadecimal representation.
MCB	Extract only alphabetic and numeric characters.
MC/B	Extract only special characters that are neither alphabetic nor numeric.
MCBA	Convert binary representation to ASCII characters.
MCBX	Convert a binary value to its hexadecimal equivalent.
MCC;x;y	Change all occurrences of character string x to character string y.
MCDR	Convert a decimal value to its equivalent Roman numerals. Input conversion is effective.
MCDX or MCD	Convert a decimal value to its hexadecimal equivalent. Input conversion is effective.
MCL	Convert all upper case letters (A-Z) to lower case.

MCN	Extract only numeric characters (0-9).
MC/N	Extract only non-numeric characters.
MCNP{c}	Convert paired hexadecimal digits preceded by a period or character c to ASCII code.
MCP{c}	Convert each non-printable character (X”00” - X”IF”, X”80” - X”FE”) to a period (.) or to character c.
MCPN{c}	Same as MCP but insert the two-character hexadecimal representation of the character immediately after the period or character c.
MCRD or MCR	Convert Roman numerals to the decimal equivalent. Input conversion is effective.
MCT	Convert all upper case letters (A-Z) in the text to lower case, starting with the second character in each word. Change the first character of each word to upper case if it is a letter.
MCU	Convert all lower case letters (a-z) to upper case.
MCXA or MY	Convert hexadecimal representation to ASCII characters.
MCXB{S}	Convert a hexadecimal value to its binary equivalent. Use S to suppress spaces between each block of 8 bytes.
MCXD or MCX	Convert a hexadecimal value to its decimal equivalent. Input conversion is effective.

Changing Case

Use the following MC codes to transform text from upper to lower case and visa versa are:

MCL	Convert all upper case letters (A-Z) to lower case
MCT	Convert all upper case letters (A-Z) in the text to lower case, starting with the second character in each word. Change the first character of each word to upper case.
MCU	Convert all lower case letters (a-z) to upper case.

Input conversion does not invert. The conversion code will be applied to the input data.

EXAMPLE 1

```
MCL
```

Assuming a source value of AbCdEf, MCL will return abcdef.

EXAMPLE 2

```
MCT
```

Assuming a source value of AbC dEf “ghi, MCT will return Abc Def “ghi.

EXAMPLE 3

```
MCU
```

Assuming a source value of AbCdEf, MCU will return ABCDEF.

Extracting Characters

Use the following MC codes to extract characters from a string:

MCA	Extract only alphabetic characters.
MC/A	Extract only non-alphabetic characters.
MCB	Extract only alphanumeric characters.
MC/B	Extract only special characters that are neither alphabetic nor numeric.
MCN	Extract only numeric characters (0-9).
MC/N	Extract only non-numeric characters.

Input conversion does not invert. The original code will be applied to input data.

EXAMPLE 1

```
MCA
```

Assuming a source value of ABC*123!DEF, MCA will return ABCDEF.

EXAMPLE 2

```
MC/A
```

Assuming a source value of ABC*123!DEF, MC/A will return *123!

EXAMPLE 3

```
MCB
```

Assuming a source value of ABC*123!DEF, MCB will return ABC123DEF.

EXAMPLE 4

```
MC/B
```

Assuming a source value of ABC*123!DEF, MC/B will return *!

EXAMPLE 5

```
MCN
```

Assuming a source value of ABC*123!DEF, MCN will return 123.

EXAMPLE 6

```
MC/N
```

Assuming a source value of ABC*123!DEF, MC/N will return ABC*!DEF

Replacing Characters

Some MC codes replace one set of characters with other characters. These codes can:

- Exchange one character string for another
- Replace non-printable characters with a marker character
- Replace non-printable characters with a marker character and the character's hexadecimal representation
- Replace the marker and hexadecimal representation with the ASCII code

MCC;x;y	Change all occurrences of character string x to character string y.
MCP{c}	Convert each non-printable character (X"00" - X"IF", X"80" - X"FE") to character c, or

	period (.) if c is not specified.
MCPN{c}	Same as MCP but insert the two-character hexadecimal representation of the character immediately after character c, or tilde (~) if c is not specified.
MCNP{c}	Convert paired hexadecimal digits preceded by a tilde or character c to ASCII code. The opposite of the MCPN code.

Input conversion does not invert. The original code will be applied to input data.

EXAMPLE 1

```
MCC;X5X;YYY
```

Assuming a source value of ABC*X5X!DEF, MCC will return ABC*YYY!DEF.

EXAMPLE 2

```
MCPN
Assuming a source value of ABC]]DEF where ] represents a value mark,
MCPN will return ABC.FC.FCDEF.
```

Converting Characters

The MC codes that convert ASCII character codes to their binary or hexadecimal representations or vice versa are:

MCAB{S}	Convert ASCII character codes to binary representation (Use S to suppress spaces).
MCAX or MY	Convert ASCII character codes to hexadecimal representation
MCBA	Convert binary representation to ASCII characters.
MCXA or MX	Convert hexadecimal representation to ASCII characters.

Comments: The MCAB and MCABS codes convert each ASCII character to its binary equivalent as an eight-digit number. If there is more than one character, MCAB puts a blank space between each pair of eight-digit numbers. MCABS suppresses the spaces.

When converting from binary to ASCII characters, MCBA uses blank spaces as dividers, if they are present. MCBA scans from the right-hand end of the data searching for Elements of “eight-bit” binary strings. If it encounters a space and the element is not eight binary digits long, it prepends zeros to the front of the number until it contains eight digits and continues until reaching the leftmost digit prepending zeros if necessary, it then converts each eight-digit element to its ASCII character equivalent.

Input conversion does not invert. The original code will be applied to input data.

EXAMPLE 1

```
MCAX
```

Assuming a source value of ABC, MCAX will return 414243.

EXAMPLE 2

```
MCXA
```

Assuming a source value of 414243, MCXA will return ABC.

EXAMPLE 3

```
MCAB
```

Assuming a source value of AB, MCAB will return 01000001 01000010.

EXAMPLE 4

```
MCABS
```

Assuming a source value of AB, MCABS will return 0100000101000010.

EXAMPLE 5

```
MCBA
```

Assuming a source value of 01000001 1000010, MCBA will return AB. Note the missing binary digit at the start of the second element of the source value.

EXAMPLE 6

```
MCBA
```

Assuming a source value of 0100000101000010, MCBA will return AB.

Converting Numeric Values

The MC codes that convert numeric values (as opposed to characters), to equivalent values in other number schemes are:

MCBX{S}	Convert a binary value to its hexadecimal equivalent. Use S to suppress spaces.
MCDR	Convert a decimal value to its equivalent Roman numerals. Input conversion is effective.
MCDX or MCD	Convert a decimal value to its hexadecimal equivalent. Input conversion is effective.
MCRD or MCR	Convert Roman numerals to the decimal equivalent. Input conversion is effective.
MCXB{S}	Convert a hexadecimal value to its binary equivalent. Use S to suppress spaces.
MCXD or MCX	Convert a hexadecimal value to its decimal equivalent. Input conversion is effective.

Comments: These codes convert numeric values rather than individual characters. For Example, conversion of the decimal value of 60 is to X”3C” in hexadecimal, or LX in Roman numerals. The value 60 is converted, not the characters “6” and “0”.

With the exception of MCBX{S} that handles spaces, all conversion of these codes will stop if they encounter an invalid character that is not a digit of the source number system.

With the exception of MCDR, if the conversion fails to find any valid digits, a zero MCDR will return null.

If you submit an odd number of hexadecimal digits to the MCXB code, it will add a leading zero (to arrive at an even number of characters) before converting the value.

The MCXB and MCXBS codes convert each pair of hexadecimal digits to its binary equivalent as an eight-digit number. If there is more than one pair of hexadecimal digit, MCXB puts a blank space between each pair of eight-digit numbers. MCXBS suppresses the spaces.

When converting from binary to hexadecimal digits, MCBX uses blank spaces as dividers if they are present. MCBX effectively scans from the right-hand end of the data searching for Elements of eight-bit binary digits. If it encounters a space and the element is not a multiple of eight binary digits, it prepends zeros to the front of the number until it contains eight digits. This continues until it reaches the leftmost digit prepending zeros if necessary. Each eight-digit element is converted to a hexadecimal character pair.

Input conversion is effective for MCDR, MCDX, MCRD and MCXD. Input conversion is not inverted for the other codes. The original code will be applied to input data.

EXAMPLE 1

```
MCBX
```

Assuming a source value of 01000001 1000010, MCBX will return 4142. Would return the same value if there was no space between the binary source Elements.

EXAMPLE 2

```
MCRD
```

Assuming a source value of MLXVI, MCRD will return 1066.

EXAMPLE 3

```
MCDX
```

Assuming a source value of 1066, MCDX will return 42A.

MD CONVERSION

The MD code transforms integers by scaling them and inserting symbols, such as a currency sign, thousands separators, and a decimal point. The ML and MR codes are similar to MD but have greater functionality.

COMMAND SYNTAX

MDn{m}{Z}{,}{$}{ix}{c}

SYNTAX ELEMENTS

n	a number from 0 to 9 that specifies how many digits are to be output after the decimal point; inserts trailing zeros as necessary. If n is omitted or 0, the decimal point is not output.
n	a number from 0 to 9, which represents the number of digits that the source value contains to the right of the implied decimal point. Uses m as a scaling factor and the source value is descaled (divided) by that power of 10. For Example, if m=1, the value is divided by 10; if m=2, the value is divided by 100, and so on. If m is omitted, it is assumed equal to n (the decimal precision). If m is greater than n, the source value is rounded up or down to n digits. The m option must be present if the ix option is used and both the Z and $ options are omitted. This to remove ambiguity with the ix option.
z	suppresses leading zeros. Note that fractional values, which have no integer, will have a zero before the decimal point. If the value is zero, a null will be output.
,	specifies insertion of the thousands separator symbol every three digits to the left of the decimal point. The type of separator (comma or period) is specified through the SET THOU Command. (Use the SET DEC Command to specify the decimal separator.)
$	appends an appropriate currency symbol to the number. The currency symbol is specified through the SET MONEY Command.
ix	aligns the currency symbol by creating a blank field of “i” number of columns. The value to be output overwrites the blanks. The “x” parameter specifies a filler character that can be any non-numeric character, including a space.
c	appends a credit character or encloses the value in angle brackets (>). Can be any one of the following:
-	Appends a minus sign to negative values; a blank follows positive or zero values.
C	Appends the characters CR to negative values. Two blanks follow positive or zero values.

Input Conversion: works with a number that has only thousands separators and a decimal point.

EXAMPLES

MD Code	Source Value	Returned Value
MD2	1234567	12345.67
MD2,	1234567	12,345.67
MD2,$	1234567	$12,345.67
MD2,$12*	1234567	$**12,345.67
MD2,$12-	-1234567	$12,345.67-
MD42Z,$	001234567	$12,345.6700
MD42Z,$15 <	-001234567	<$ 12,345.6700>

Mk Conversion

The MK code allows you to display large numbers in a minimum of columns by automatically descaling the numbers and appending a letter to represent the power of 10 used. The letters and their meanings are:

K 10 /3 (Kilo)

M 10 /6(Mega)

G 10 /9 (Giga)

COMMAND SYNTAX

MKn

SYNTAX ELEMENTS

n represents the field width and if present will include the letter and a minus sign.

Comments: will not change if a number will fit into the specified field width.

If the number is too long but includes a decimal fraction, the MK code first attempts to round the fractional part so that the number will fit the field. If the number is still too long, the code rounds off the three low-order integer digits, replacing them with a K. If the number is still too long, the code rounds off the next three digits, replacing them with an M. If that is still too long, the code rounds off three more digits, replacing them with a G. If the number still does not fit the specified field, the code displays an asterisk. If the field size is not specified or is zero, the code outputs null.

Input Conversion: does not invert. It simply applies the metric processing to the input data.

EXAMPLES

Source Data	MK3	MK4	MK5	MK7
1234	1K	1234	1234	1234
123456789	*	123M	123M	123457K
123456789012345	*	*	*	123457G
999.9	1K	1000	999.9	999.9
-12.343567	-12	-12	-12.3	-12.344
-1234.5678	-1K	-1K	-1235	-1234.6
-0.1234	-.1	-.12	-.123	-0.1234

Ml/MR Conversion

ML and MR codes format numbers and justify the result to the left or right respectively. The codes provide the following capabilities:

- Decimal precision and scaling
- Zero suppression
- Thousands separator
- Credit codes
- Currency symbol
- Inclusion of literal character strings

COMMAND SYNTAX

ML{n{m}}{Z}{,}{c}{$}{fm}
MR{n{m}}{Z}{,}{c}{$}{fm}

SYNTAX ELEMENTS

ML	Provides left justification of the result.
MR	Provides right justification of the result.
n	a number from 0 to 9 that defines the decimal precision. It specifies the number of digits to for output following the decimal point. The processor inserts trailing zeros if necessary. If n is omitted or is 0, a decimal point will not be output.
m	a number that defines the scaling factor. The source value is descaled (divided) by that power of 10. For Example, if m=1, the value is divided by 10; if m=2, the value is divided by 100, and so on. If m is omitted, it is assumed equal to n (the decimal precision).
z	suppresses leading zeros. Note that fractional values, which have no integer, will have a zero before the decimal point. If the value is zero, a null will be output.
,	the thousands separator symbol. It specifies insertion of thousands separators every three digits to the left of the decimal point. You can change the display separator symbol by invoking the SET THOU Command. Use the SET DEC Command to specify the decimal separator.
C	Print the literal CR after negative values.

D	Print the literal DB after positive values.
E	Enclose negative values in angle brackets < >
M	Print a minus sign after negative values.
N	Suppresses embedded minus sign.

If a value is negative and you have not specified one of these indicators, displays the value with a leading minus sign. If you specify a credit indicator, outputs the data with either the credit characters or an equivalent number of spaces, depending on its value.

$	Specifies that a currency symbol is to be included. Places a floating currency symbol in front of the value. The currency symbol is specified through the SET MONEY Command.
Fm	Specifies a format mask. A format mask can include literal characters as well as format codes. The format codes are as follows:
Code	Format
#{n}	Spaces. Repeat space n times. Overlays the output value on the spaces created.
*{n}	Asterisk. Repeat asterisk n times. Overlays the output value on the asterisks created.
%{n}	Zero. Repeat zeros n times. Overlays the output value on the zeros created.
&x	Format. x can be any of the above format codes, a currency symbol, a space, or literal text. The first character following '&' is used as the default fill character to replace #n fields without data. You may enclose format strings enclosed in parentheses "( )".

Comments: The justification specified by the ML or MR code applies at different stages from that specified in field 9 of the data definition record. The sequence of events begins with the formatting of the data with the symbols, filler characters and justification (left or right) specified by the ML or MR code. The formatted data is justified according to field 9 of the definition record and overlaid on the output field, which initially comprises the number of spaces specified in field 10 of the data definition record.

Input Conversion: works with a number that has only thousands separators and a decimal point.

EXAMPLES

Conversion Code	Source Value	Dict Fields 9 and 10	Returned Value (columns) 12345678901234567890

MR2#10	1234	L 15	12.34
MR2#10	1234	R 15	12.34
ML2%10	1234	L 15	12.3400000
MR2%10	1234	R 15	0000012.34
ML2*10	1234	L 15	12.34*****
MR2*10	1234	R 15	*****12.34
MR2,$#15	12345678	L 20	#123,456.78
MR2,&$#15	12345678	L 20	#####123,456.78
ML2,&*$#15	12345678	L 20	#123,456.78*****
MR2,& $#15	12345678	L 20	# 123,456.78
MR2,C&*$#15	-12345678	L 20	#***123,456.78CR
ML& ###-##-###	123456789	L 12	123-45-6789
ML& #3-#2-#4	123456789	L 12	123-45-6789
ML& Text #2-#3	12345		Text 12-345
ML& ((Text#2) #3)	12345		(Text12) 345

In the last Example, it ignores the leading and trailing parenthesis.

MP Conversion

MP codes convert packed decimals to unpacked decimal representation for output or decimal values to packed decimals for input.

COMMAND SYNTAX

MP

Comments: The MP code most often used as an output conversion; on input, the MP processor combines pairs of 8-bit ASCII digits into single 8-bit digits as follows:

a. Strips off the high order four bits of each ASCII digit.
b. Moves the low order four bits into successive halves of the stored byte

c. Adds a leading zero (after the minus sign if present) if the result would otherwise yield an uneven number of halves.
d. Ignores leading plus signs (+)
e. Stores leading minus (-) signs as a four-bit code (D) in the upper half of the first internal digit.

When displaying packed decimal data, you should always use an MP or MX code. Raw packed data is almost certain to contain control codes that will upset the operation of most terminals and printers.
Input Conversion: is valid. Generally, for selection processing you should specify MP codes in field 7 of the data definition record.

EXAMPLES

```
OCONV -1234 "MP"
yields 0x D01234
ICONV 0x D01234 "MP"
yields -01234
```

MS Conversion

The MS code allows an alternate defined sort sequence for sort fields.

COMMAND SYNTAX

MS

Comments: Use of the MS code is only relevant when applying in field 8 pre-process codes to a specified field in a sort clause. In all other cases, it will be ignored.

Use the sort sequence defined in a special record named SEQ that you must create in the ERRMSG file. Field 1 of this record contains a sequence of ASCII characters that define the order for sorting.

EXAMPLE

```
SEQ (defined in ERRMSG file)
001 aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyY
zZ9876543210 ,.?!"";:+-*/^=()[]{}<>@#$%&"~\|
INV.CODE (data definition record)
001 A
008 MS
SORT SALES BY INV.CODE ID-SUPP
SALES....
AbC789
ABC789
ABC788
dEF123
```

MT Conversion

Use the MT code to convert time notations such as 01:40:30 or 1:30 AM between internal and external format.

COMMAND SYNTAX

MT{H}{S}

SYNTAX ELEMENTS

H specifies 12-hour format. Uses 24-hour format if omitted
S specifies that seconds are to be included.

Comments: Time is stored internally as the number of seconds since midnight. Outputs the stored value in 12 hour or 24 hour (international) format
Defines 12:00PM as noon and 12:00AM as midnight
Automatically displays AM and PM designators. For Example: 09:40AM and 06:30PM.

Input Conversion: is valid. Generally, for selection processing you should specify MT codes in field 7 of the data definition record.
Considers AM or PM designators; affects the result of the input conversion for certain values by the time_is_hours emulation setting.

EXAMPLES

Input Conversion

Code	Input	Result
MT	00:00	0
MTH	12:00AM	0
MT	01:00AM	3600
MT	01:00	3600
MTH	01:00	3600
MTH	01:00AM	3600

MT	01:00PM	46800
MTH	01:00PM	46800
MTS	01:00:30	3630

Output Conversion

Code	Source Value	Result
MTS	0	00:00:00
MTHS	0	12:00:00AM
MT	3600	01:00
MTH	3600	01:00AM
MT	46800	13:00
MTS	46800	13:00:00
MTH	46800	01:00PM
MTHS	46800	01:00:00PM

P Conversion

The P code returns a value if it matches one of the specified patterns, which can be combinations of numeric and alphabetic characters and literal strings.

COMMAND SYNTAX

P{#}(element){;(element)}...

SYNTAX ELEMENTS

#	Returns null if the source matches the pattern, or the source if it does not.

Element is one or more of the following:

nA	tests for n alphabetic characters
nC	tests for n alphabetic or numeric characters
nN	tests for n numeric characters
nP	tests for n printable characters
nX	tests for n characters
“literal”	tests for the presence of the literal

Comments: Returns a null value if the value does not match any of the patterns.

Input Conversion: does not invert. It simply applies the pattern matching to the input data.

EXAMPLE 1

P(2A”*”3N”/”2A)

Will match and return AA*123/BB or xy*999/zz. Will fail to match AAA*123/BB or A1*123/BB, and will return null.

EXAMPLE 2

P(2A”*”3N”/”2A);(2N”-“2A)

Will match and return AA*123/BB, xy*999/zz, 99-AA or 10-xx. Will fail to match AA&123/BB, A1*123/BB, 9A-AA or 101-xx, and will return null.

R Conversion

The R code returns a value that falls within one or more specified ranges.

COMMAND SYNTAX

Rn,m{;n,m}...

SYNTAX ELEMENTS

n	the starting integer of the range. Can be positive or negative.

m the ending integer of the range. Can be positive or negative, but must be equal to or greater than n.

Comments: Returns a null value if the value does not fall within the range(s).

Input Conversion: does not invert. It simply applies the range check to the input data.

EXAMPLE 1

R1,10

Will return any value that is greater than or equal to one and less than or equal to 10.

EXAMPLE 2

R-10,10

Will return any value that is greater than or equal to -10 and less than or equal to 10.

EXAMPLE 3

R-100,-10

Will return any value that is greater than or equal to -100 and less than or equal to -10.

S Conversion

The S code substitutes one value for another.

COMMAND SYNTAX

S;Var1;Var2

SYNTAX ELEMENTS

Var1 specifies the value to be substituted if the referenced value is not null or zero. Can be a quoted string, an FMC (field number), or an asterisk. An asterisk indicates that you should use the value of the referenced field.

Var2 specifies the value for substitution if the referenced value is null or zero. Can be a quoted string, an FMC (field number), or an asterisk.

EXAMPLE 1

S;*;”NULL VALUE!”

If the referenced field is null, this Example will return the string “NULL VALUE!”. Else, it will return the referenced value.

EXAMPLE 2

S;*;3

If the referenced field is null, this Example will return the content of field 3 of the data record. Else, it will return the referenced value.

EXAMPLE 3

S;4;5

If the referenced field is null, this Example will return the content of field 5 of the data record. Else, it will return the content of field 4.

T Conversion

The T code extracts a character substring from a field value.

COMMAND SYNTAX

T{m,}n

SYNTAX ELEMENTS

m	specifies the starting column number.
n	The number of characters for extraction.

Comments: If specifying **m**, the content of field 9 of the data definition record has no effect - it counts and extracts characters from left to right, for n characters.
If m is not specified, the content of field 9 of the data definition record will control whether n characters are extracted from the left or the right-hand end of the value. If field 9 does not contain an R, extracts the first n characters from the value. If field 9 does contain an R (right justify), extracts the last n characters from the value.
Input Conversion: does not invert. It simply applies the text extraction to the input data.

EXAMPLES

Code	Source Value	Field9	Result
T3,4	ABCDEFG	L	CDEF
T3,4	ABCDEFG	R	CDEF
T2	ABCDEFG	L	AB
T3	ABCDEFG	R	EFG
T3	ABCDEFG	T	ABC

Tfile Conversion

Tfile codes provide a method for retrieving data fields from any other file to which the user has access.

COMMAND SYNTAX

```
T[*|DICT]file-specifier;c{n};{i-fmc};{o-fmc}
```

SYNTAX ELEMENTS

* or DICT	Indicates the use of the dictionary of the specified file, rather than the data section.
file-specifier	identifies the reference file by name in the format file-name{,data-section-name}.
c- specifies a translation code, which can be any one of the following:	
C	If reference record does not exist or the specified FMC is null, output the value unchanged.
I	Inputs verify: Functions as a C code for output and as a V code for input.
O	Outputs verify: Functions as a C code for input and as a V code for output.
V	Reference record must exist and the specified FMC must contain a translatable value. If the record does not exist or the FMC contains a null, an error message will be output.
X	If reference record does not exist or the specified FMC is null, return a null
n	specifies a value mark count to return one specific value from a multivalued field. Returns all values if omitted.
i fmc	the field number for input translation. which if omitted or contains a null value, no input translation takes place.
o fmc	is the field number for output translation. If the value is null, no output translation takes place.

Comments: Uses the current data value as the record key for searching the specified reference file. Returns a data field or a single value from a data field, from the record

Use Tfile codes in fields 7 or 8 of the data definition record. Use field 8 if translation of a multivalued field or comparisons and sorts are required.

If you apply selection criteria, you can either use field 8, or field 7 and set up special records in the reference file to perform any input translation you require.

The special records in the reference file have as record keys values that the field subject to translation may be compared with in a jQL sentence. Field i-fmc within these records contains the translate value that will be compared to values on file. Typically, values in a jQL sentence are output values, so that the special input translation records are effectively the inverse of the output translation records.

Tfile codes can be “embedded” in other conversion codes but you must still follow the syntactical conventions of the “host” code. For Example, if you include a Tfile code in an F code conversion, enclose the Tfile code in parentheses.

Output conversion is valid. The Tfile code has a parameter (o-fmc) that specifies the field in the translation record to use for output conversion.

Input conversion is valid. The Tfile code has a parameter (i-fmc) that specifies the field in the translation record to use for input conversion.

EXAMPLE 1

TSALES;X;;2

Using this Tfile code in field 8 of a data definition record, which also has a 0 in field 2, will cause the key of the current record to be used as the key when accessing the reference file SALES; returns null if the record cannot be found; returns the value of field 2 if the record is found.

EXAMPLE 2

TSALES;C;;2

Using this Tfile code in field 8 of a data definition record, which also has a 6 in field 2, will cause the content of field 6 from the current record to be used as the key when accessing the reference file SALES. If the record cannot be found, or if found, field two is null, returns the content of field 6 of the current record. If the record is found, and field 2 contains a value, it returns that value.

EXAMPLE 3

A;3(TSALES;X;;2)

Using this embedded Tfile code in field 8 of a data definition record will cause the use of field 3 of the current record as the key when accessing field 2 of the reference file SALES. Returns null if the record cannot be found; returns the value of field 2 if the record is found.

U Conversion

Use the U code to execute a system subroutine to process values.

COMMAND SYNTAX

Uxxxx

SYNTAX ELEMENTS

XXXX The hexadecimal identity of the routine.

Comments: jBASE user exits are customized routines specially produced to perform extraordinary processing.

Input Conversion: Routine dependent

Chapter 6 Default Data Definition Records

When issuing a jQL Command without containing specific references to data definition records, nor do you suppress the output of the report detail, the system will attempt to locate any default data definition records, which may be set up.

For Example: if you issue the Command **"LIST SALES"**, the system will look in the dictionary of the SALES file for a data definition record named "1". If it finds "1", this will become the default output for column two. The system will then look for a data definition record named "2" and so until the next data definition record is not found. If "1" is not found in the file dictionary, the system will search the default dictionaries for the same sequence of data definition records.

When you issue a jQL Command, which does not contain specific references to data definition records, the system will first attempt to locate each data definition record in the dictionary of the file (or in the file specified in a USING clause). If no data definition is found in the dictionary (or the file specified in a USING clause), the system will look for the data definition in the file defined by the JEDIFILENAME_MD environment variable.

For Example: if you issue the Command "LIST SALES VALUE", the system will look in the dictionary of the SALES file for a data definition record named "VALUE". If it cannot find "VALUE" in the file dictionary, the system will look in the file specified by the JEDIFILENAME_MD environment variable. In this way, you can set up data-specific, file-specific or account-specific defaults for use with any jQL Command.

jQL Output (Reports)

By default, displays output from a jQL Command on your terminal, in columnar format, with a pause at the end of each page (Full screen).

Output Device

You can redirect the output to a printer (or the currently-assigned Spooler device) by using the LPTR format specifier or the P option.

Report Layout

If the columnar report will not fit in the current page width of the output device, it will be output in “non-columnar” format where each field of each record occupies one row on the page.

Paging

If the displayed report extends over more than one screen, press <ENTER> to view the next screen. To exit the report without displaying any remaining screens, press <Control X> or “q”

jQL Basic Subroutines

jBASE jQL enables users to call Basic subroutines from within correlatives and conversions. There are two flavors of subroutine and each requires a different include file. For Advanced Pick subroutines, the developer must include the following header file from the “include” subdirectory in the jBASE release directory.

qbasiccommonpick

For Sequoia subroutines, the developer must include the following header file from the “include” subdirectory in the jBASE release directory.

qbasiccommonseq

File Definitions

This section starts with a review of file definition records and discusses how the content of these records can affect the output from jQL Commands - particularly with reference to sublists. File definition records contain information used when formatting reports.

For Example: left or right justified, and how wide a column should it occupy. You can also define sublist codes to tell the system that a particular field is multivalued.

Record Structure

The fields of a file definition record that affect jQL reports are:	
Field 7	Conversion code for key if required. For date, time, etc.,
Field 8	V code to notify a multivalued (sublist) field if required. See sublists - V code.
Field 9	Justification for key. Can be one of the following (see data definition records) L Left justified R Right justified T Text U unlimited
Field 10	Column width for Key. Default is 14 characters.

Sublists - V Code

You can access file records, which contain sublists, with the COUNT and LIST Commands. For the commands and the modifier to function correctly, you must include the V processing code in field 8 of the file definition record. See File Specifiers topic in the jQL Sentence Construction Chapter for more details.

COMMAND SYNTAX

V;field-no

SYNTAX ELEMENTS

Field No. The number of the field, which contains the sublist

EXAMPLE

Consider the stock file used by a camera factory where each data record can represent either an assembly or a component part. Take as an Example the record set that defines a simple camera assembly. The data records contain the following data.

Key	A1	Key	A21
001	CAMERA ASSY	001	LENS ASSY
002	A21]A22]A23	002	A210]A211
003	10	003	15

Key	A22	Key	A23
001	BODY	001	SHUT ASSY
002		002	A230]A231
003	10	003	11

Key	A210	Key	A211
001	OPTICS	001	BARREL
002		002	
003	19	003	21

Key	A230	Key	A231
001	IRIS MECH	001	IRIS HOUSNG
002		002	
003	13	003	14

The key is the part number, field 1 contains the description, field 2 is a multivalued list of components that go to make up the part, and field 3 is the current stock level.
Record A1 represents assembled cameras. It points to the used sub-assemblies (A21, A22 and A23) to make each camera. The sub-assemblies in turn point to their component parts; A21 points to A210 and A211, A22 does not have any components, and A23 points to A230.
Having established the logical data relationships, we now need to ensure that the system understands that field 2 is a multivalued sublist. We do this by updating field 8 in the file definition record to read "V;;2",
like this:

```
STOCK
001 D
002
003
004
005
006
007
008 V2
009 L
010 10
```

To create three data definition records in the dictionary of STOCK - one for each field, use the following titles DESC, COMPONENTS, and QTY.

The final step is to issue a COUNT or LIST Command which uses the WITHIN modifier:

```
LIST WITHIN STOCK "A1" DESC COMPONENTS QTY
```

```
PAGE 1 Time Date

LEVEL STOCK Description.... Components Qty

1 A1 CAMERA ASSY A21 10
A22
A23
2 A21 LENS ASSY A210 15
A211
3 A210 OPTICS 19
3 A211 BARREL 21
2 A22 BODY 10
2 A23 SHUTTER ASSY A230 11
A231
3 A230 IRIS MECH 13
3 A231 FILM MECH 14

8 RECORDS LISTED
```

Chapter 7 Data Definition Records

Data definition records (sometimes known as field definition records) define the characteristics of each field in a data file. They specify the output format and the type of processing required to generate each column of a jQL report.
Use data definition records to:

a. Specify default output.
b. Associate field names with field numbers (column headings).
c. Perform output formatting.
d. Calculate new values based on the source data
e. Perform processing via conversion codes.

Although normally used to define a single physical field in a file, use the data definition records for operations that are more complex. For Example:

- To “join” or derive data from other fields or files
- To verify the presence (or absence) of records in other files
- To format their output in the most easily understood manner (to convert numeric 0 and 1 flags to “Yes” or “No”, for Example, or to output text like “Overdue” if one date field is older than another).
- To generate statistical data like record sizes or counters

The data definition records are usually located in the dictionary of the data file (but not always - see the USING Clause and the Default Output Specification topics). You can set up any number of data definition records. Often, there are several definitions for each field, each one used by a different set of reports which have different output requirements.
You associate the data definition record with a particular field in the data file by specifying the target fields FMC (field-mark count) in field 2 of the data definition record. The FMC refers to (points to) the field number (also known as the line number) of the data within the records of the data file.

Record Layout

All data definition records are defined in the same way:
Field|Description

7. D/CODE|Defines the record as a data definition record. Must be one of the following codes:

A	Marks a normal data definition record.

S	Obsolete but still supported. Was like the A type, but suppressed default column headings when field 3 was blank. Replaced by the A type with a backslash in field 3 to defeat heading.
X	Forces the definition to be ignored if selected as part of a default set of data definitions. Use only when explicitly named. See Default Output Specification later.

8. FMC (field-mark count)|A field number or special FMC (see Special Field-mark Counts for more details). A field number refers to the corresponding field (or line) in a record.

The special FMC codes are:

0	Refers to the record key - field number 0 (zero).
9998	Ordinal number of record at output (used for numbering or counting).
9999	Size of the record in bytes (excluding the key).

9. Column heading|
    - Heading text, null, or a backslash followed by text.
    - Entering more characters here than the width in field 10 allows, increases the width to accommodate the heading text (this field wins).
    - If the statement uses the COL-HDR-SUPP output modifier or the “C” option it displays no column headings.
    - Uses heading text as the column heading. If the text is less than the column width, it will be padded with dots.
    - Use spaces to produce a blank heading.
    - Use value marks, (ctrl]), as NEWLINE characters to place the following text on a new line. If this field is left null, uses the key of the data definition record as the column heading.
    - Text following a backslash “\” character will be used as the column heading. If nothing follows the backslash, the column heading will be null.
10. 4 - 6|Not used.

11. Input/Output conversion codes|Used for processing the data after sort and selection but before output. See Conversion Codes. Multiple conversion codes, separated by a value marks, will be processed from left to right.
12. Pre-process conversion codes|Used for processing the data before sort and selection and before field 7 codes. See Conversion Codes later. Multiple conversion codes, separated by a value marks, will be processed from left to right.
13. Format|Specifies the layout of the data within the column. Can be any of the following:

L Left justified	If the data exceeds the column width specified in field 10, the data is broken at column width without respect to blank spaces.
R Right justify	If the data exceeds the column width specified in field 10, it truncates the data.
T Text. Word wrap - like L	(Left justified) Where possible, lines will be broken at the blank space between words.
U Unlimited.	Data is output on one line ignoring column boundaries.

10. Width|Numeric value specifying the column width; If the number of characters in field 3 (the heading) is greater than the number entered here, the number of characters in field 3 will be used.

Special Field-mark Counts

Three special FMCs (field-mark counts) are recognized: 0, 9998 and 9999.

FMC 0 - RECORD KEY

Setting field 2 of the data definition record to 0 (zero) causes the system to work with the record key. In this way, you could set up a data definition record which would allow a the record keys to be output in a column other than the first, and to use any column heading.

Typically, you would also use the ID-SUPP modifier or the “I” Command option to suppress output of the record key in the first column.

FMC 9998 - RECORD COUNT/NI OPERAND

Setting field 2 of the data definition record to 9998 causes the system to return a record (or line) count equal to the number of records output so far in the report.

You could also use function operators within an A or F conversion code in field 7 or 8 of the data definition record to achieve the same result. Function code operand NI yields the same value as an FMC of 9998.

FMC 9999 - RECORD SIZE/NL OPERAND

Setting field 2 of the data definition record to 9999 causes the system to return the record size in bytes. The size does not include the key but does include all field marks within the record.
You could also use function operators within an A or F conversion code in field 7 or 8 of the data definition record to achieve the same result. Function code operand NL yields the same value as an FMC of 9999.

Default Output Specification

Default output specifications work in two ways, depending on whether the default definitions are explicit or implicit.

Explicit Defaults

If you specify and use a data definition record for output, the system will search the implied dictionary first (or the dictionary specified in a USING clause). If the data definition is not found in the implied dictionary, the system will look for the data definition in the file defined by the JEDIFILENAME_MD environment variable.

Implicit Defaults

If you do not explicitly specify or use any data definition records for output, the system will search for a predefined series of records based on the search criteria outlined in the preceding section.
You can therefore set up a series of data definition records, which the system will use if a jQL Command sentence does not include any explicit output file, Ids.
You must name these “default” records in a numeric sequence starting at 1 (1, 2, 3, and so on). The fields, which these records define, will be output in the same sequence as the keys but they do not need to follow the same sequence as the fields in the file.
When a jQL Command sentence with no explicit output fields is issued, the system first looks in the dictionary for a data definition record named 1, then for a record named 2, then 3, and so on until it fails to find a record with the next number. Will use the record if it has a D/CODE of A; it ignores the record if it has a D/CODE of X, but it will not break the sequence.
Will skip a record with a D/CODE of X if it was found as the result of a search for defaults; Under normal circumstances it can be used in the same way as any other data definition record.
This means that when you first set up a series of “default” data definition records, you should put an A in the D/CODE field of each. If you subsequently need to remove one from the sequence, you can simply change the D/CODE field to an X. This way you do not break the sequence or have to copy the remaining “default” records to new names in order to fill the gap.

You can still use a data definition record with a number for a key in the same way as any other data definition record.

Predefined Data Definition Records

Some predefined data definition records are automatically available so that, if appropriate data definition records are not included in a files dictionary, you can still generate a report. These records are recognized when used in a jQL Command sentence.

The predefined data definition records are named *A0 to *Annn. The numeric portion of the key corresponds to the position of the field they report on and the column heading will be the same as the DDR name.

I-TYPES

The jBASE jQL processor supports I-TYPES as imported from PRIME or Universe.

The jBASE query language, jQL, has been enhanced to support D and I type attribute definition records.

Formats

```
I-TYPE
001 I
002 Expression
003 Conversion
004 Header
005 Format
006 - 016 Reserved

D-TYPE
001 D
002 AttributeNo
003 Conversion
004 Header
005 Format
006 - 016 Reserved
```

Expression

This can be one or more of the following types

Type	Description
Dictionary Id	E.g. Attribute definition S/A/D/I/X/V type (Note: V is equivalent to I)
@Variables	E.g. @RECORD, @ID, @USERNO, @DATE, @TIME
Functions	E.g. TRANS (File, Item, Attr, Code)
User Subroutines	E.g. MYSUB (param1, param2, param256)
Conditionals	E.g. IF X = Y THEN @RECORD ELSE " "
String Extraction	E.g. Expression[6,4]

You can define multiple expressions within the same I-TYPE. E.g.
Expression ; Expression ;
Expressions can be parenthesized, contain numeric constants, string literals, enclosed in single or double quotes, and extended operators such as EQ, NE, LE, GT, CAT, AND, OR, MATCHES.

User Subroutines

You can add additional functionality by calling user written basic subroutines, which you should compile and catalog and add the library location to the library path in the JBCOBECTLIST environment variables.
The first parameter of the called routine is the result parameter; used as the evaluated value of the subroutine e.g.

```
FRED
001 SUBROUTINE FRED(Result, Param1)
002 IF Param1 > 100 THEN Result = 1 ELSE Result = 0
003 RETURN
```

One or other of the following formats can call subroutines from an I-TYPE.
FRED(param1 {,param2_}) or SUBR("FRED",param1 {,param2_})

Conversion

The Conversion attribute provides support for normal queries output conversions. E.g. D2, MT, F;, TFile etc

Header

This attribute specifies the column heading text for display.

Format

The format attribute can be specified as follows:

Length {Padding} Justification { Conversion } { Mask }

Where:

Length	the display column length
Padding	Any character except L,R,U or T. default space
Justification	L Left, R Right, T Text, U Unlimited

n	Number of digits after decimal point.
$	Precede with current currency sign.
,	Insert thousandths separator every third digit.
Z	Suppress leading zeroes.
Mask	Output pattern. e.g. ##-###-##

NOTE: Spurious trailing spaces can give invalid conversion errors.

ICOMP

Using an I-TYPE for the first time in a query, i.e. jQL Command, the expression attribute will be “compiled”, to produce internal op codes and parameter definitions. This mechanism provides greater efficiency at run time. However to ensure it compiles all I-TYPE definitions, rather than on an ad hoc basis, a utility, ICOMP, has been provided.
Called as:
ICOMP {DICT} FileName {RecordList | * }

Where:

FileName	The name of the file to convert.
RecordList	The list of the Record identifiers.

NOTE: ICOMP will always attempt to convert the dictionary section of a file. If RecordList is omitted, it compiles all I-TYPE definitions. ICOMP will also respect a preceding SELECT list.