# ACCESS reference manual

88A00776A02

ZEBRA
FAMILY

GENERAL AUTOMATION

## RECORD OF REVISIONS

Title: ACCESS Reference Manual

Document No. 88A00776A02

| Date | Revision Record |
|---|---|
| Mar 84 | Original Issue |
| Feb 85 | Revision A02 - Change Package (85A00516A01) |

# ACCESS
# reference manual

**88A00776A02**

## RECORD OF REVISIONS

Title: ACCESS Reference Manual

Document No. 88A00776A02

| Date | Revision Record |
|---|---|
| Mar 84 | Original Issue |
| Feb 85 | Revision A02 - Change Package (85A00516A01) |

## FOREWORD

This document is one of a family of ZEBRA reference manuals devoted to PICK processors that are on call within the PICK operating system. Before reading this document and using the processor described, it is recommended that you first become familiar with the PICK terminal control language and file structure. These subjects are thoroughly covered in 88A00782A, listed below with other documents covering PICK processors.

| Document No. | Title |
|---|---|
| 88A00757A | PICK Operator Guide |
| 88A00758A | ACCU-PLOT Operator Guide |
| 88A00759A | COMPU-SHEET Operator Guide |
| 88A00760A | Quick Guide for the PICK Operating System |
| 88A00774A | PICK Utilities Guide |
| 88A00777A | PICK SPOOLER Reference Manual |
| 88A00778A | PICK BASIC Reference Manual |
| 88A00779A | PICK EDITOR Reference Manual |
| 88A00780A | PICK PROC Reference Manual |
| 88A00781A | PICK RUNOFF Reference Manual |
| 88A00782A | Introduction to PICK TCL and FILE STRUCTURE |
| 88A00783A | PICK JET Word Processor Guide |

TABLE OF CONTENTS

## LIST OF APPENDIXES

# introduction 1

## 1.1 FORMING ACCESS INPUT SENTENCES

ACCESS is the data base portion of PICK. Access allows the user to make various types of listings and queries quickly and easily by using simple English words. It may also be used to select items from a file for use by other processors. The first part of the ACCESS manual (Section 1) describes the retrieval of data from the ACCESS processor. Entry of data into the ACCESS processor may be accomplished most efficiently by a BASIC program. Necessary information for a data entry program is included in Section 2. The final portion of ACCESS (Section 3) gives an extended technical description of the ACCESS selection processor.

ACCESS is directed by English sentences. The user types in various command sentences at the terminal to perform particular tasks. Like normal English, each ACCESS sentence contains a verb.

The ACCESS input sentence has the following general form:

```
verb {DICT} file-name {item-list} {selection-criteria}
{sort-keys} {output-specifications {print-limiters}}
{modifiers} {(options)}
```

A verb and a file-name are required; all other elements are optional. Thus, the minimum ACCESS sentence consists of a verb followed by a file-name. The verb specifies generally what type of processing will be performed on the file. For example,

```
>LIST INVENTORY [CR]
```

Facilities are provided for selecting, listing, or sorting items and for generation of certain file usage statistics. The file-name must be of one of the following standard forms:

| Form of File-Name | Example |
|---|---|
| file-name | BP |
| dict-name,data-name | PROJ,GREEN-ACRES |
| DICT file-name | DICT M/D |

Note that file names may not start with a left parenthesis ( "(" ) and may not contain commas ( , ).

The optional item-list specifies those items eligible for consideration (the absence of an item-list implies all items). An item-list consists of specifically enumerated item-ids, each enclosed within quotes ( ' or " ) or backslashes ( \ ).

Selection criteria, if present, further limit the items for output to those meeting the specified conditions. Many different specifications may be combined logically in order to select only those items meeting a certain set of criteria.

Any attribute name or the item-id may be specified as either an ascending or a descending sort key. Multiple sort keys may be specified.

Output specifications indicate which attribute-defining items in the dictionary of the file are to be used to format the listing. The user indicates exactly which values (fields) in the items (records) he wishes to see.

Print limiters suppress the listing of data not meeting certain specifications when an attribute has many values but only those values meeting a set of criteria are to be printed.

Various modifiers and/or options specify how output is listed. This includes spacing, how to handle totals, control breaks, suppression of item-ids, headings, or default messages.

A standard set of verbs, modifiers, and relational operators are supplied to each user. These special words are defined as items in the user's Master Dictionary (MD). Modifiers and relational operators (but not verbs) are reserved words. A user may define any number of synonyms for these words (and even remove the system defined entries), thereby creating his own semantics for the language. Thus, the user can rewrite the ACCESS language as long as the new language he creates follows the rules defined in the following section.

## 1.2 RULES FOR GENERATING ACCESS SENTENCES

The follow general rules apply to the use of ACCESS input sentences:

1. ACCESS input sentences are entered at the TCL level (when the system prompts with the sign ">") or a PROC may be written to perform ACCESS functions.

2. The first word of any ACCESS input sentence must be an ACCESS verb defined in the user's MD.

3. A sentence is terminated by a carriage return. A sentence may be continued to multiple lines by use of the line continuation character (control-underline), followed by a carriage return*. Additional lines will be prompted for with a colon (:) prompt.

4. The data in only one file may be directly referenced by an ACCESS sentence. Therefore, only one file-name may be used to specify the data source in an ACCESS sentence. File-names may consist of any sequence of nonblank characters and must be unique within the MD. The modifier "DICT" may be included in the sentence (just preceding the file-name) to specify operation on the dictionary section of the file, as opposed to the data section.

5. Any number of attribute names may be used in a sentence. Attribute names may consist of any sequence of nonblank characters. They must be contained in the dictionary of the file being listed, or in the file specified by the USING modifier, or in the user's MD. If the DICT modifier is used with the data-file name, then the attribute names must be in the MD or in the specified file if the USING connective is employed.

6. Any number of modifiers and relational operators, which have been predefined in the MD, may be used.

7. Verbs, file-names, attribute names, modifiers, and relational operators are delimited (separated) in an ACCESS sentence by blanks.

8. Specific items to be listed are described by their item-ids, which are enclosed in quotes or backslashes (e.g., "SC-128", ´0123´, \MD\) and must appear immediately following the file-name. Elements enclosed in single quotes (´) will be considered to be item references anywhere in the sentence.

9. Specified values are enclosed in double quotes or backslashes (e.g., "12.50", \DISCOUNT\) and apply to the previous attribute name.

*The character used to generate the continuation character may vary on some terminals. However, whatever character corresponds to the ASCII US code will perform this function.

## 1.3 AN OVERVIEW OF THE ACCESS VERBS

Each ACCESS sentence must begin with one ACCESS verb. These action-oriented English words will invoke specific ACCESS processors with the verbs specifying what is to be done to the data in the file. The common ACCESS verbs are briefly discussed below.

<u>LIST and SORT; LIST-LABEL and SORT-LABEL</u> - The LIST and SORT verbs are used to generate formatted output. LIST takes items from the file in the same order as they are stored, group for group. SORT will sort the items by item-id, or by any number of other specified sort keys. Generated output is formatted into a columnar output if possible, taking into account the maximum defined size of the specified attibutes and their associated names, along with the page width as defined by the TCL verb TERM. If more attributes have been specified than will fit across the page, a noncolumnar output is generated with the attribute names down the left side and the associated attribute values to the right. LIST-LABEL and SORT-LABEL are analogous to LIST and SORT, but allow more than one item to appear on one line of output.

<u>REFORMAT and SREFORMAT</u> - These verbs perform the same functions as the LIST and SORT verbs, respectively, except that the output is directed to another file or to tape or cartridge disk instead of to the terminal or printer. These verbs may be used to update items in a file, change the arrangement of attributes in a file, or to write tape or cartridge disk records.

<u>COUNT</u> - The COUNT verb counts the number of items meeting the conditions as specified by the combination of item-list and selection-criteria. The output generated by this verb is simply the number of items counted.

<u>SUM and STAT</u> - The SUM and STAT verbs provide a facility for summing (totaling) one specified attribute name. The STAT verb additionally provides a count and average for the specified attribute name. The output generated by these verbs are the derived statistics.

<u>SELECT, SSELECT and QSELECT</u> - The SELECT verb provides a facility to select a set of item-ids or values using the item-list and the selection-criteria. SELECT generates a list of item-ids or values; SSELECT generates a sorted list. QSELECT creates a list from attribute(s) within item(s) in a file. The list is then available to other ACCESS or TCL-II processors. The very next ACCESS or TCL-II verb executed will have access to this list, so the set of item-ids or values processed by the next verb will be those selected by the SELECT, SSELECT or QSELECT verb.

<u>SAVE-LIST, GET-LIST, and DELETE-LIST</u> - The SAVE-LIST, GET-LIST, and DELETE-LIST verbs are used to save, restore, and delete lists created by SELECT and SSELECT statements. SAVE-LIST will save a list of item-ids or values generated by a SELECT or SSELECT verb by "cataloging" the list in the POINTER-FILE. The GET-LIST verb retrieves a list from the POINTER-FILE, at which time the retrieved list is handled exactly like a list generated by a SELECT or SSELECT verb. DELETE-LIST will remove a cataloged list from the POINTER-FILE.

COPY-LIST and EDIT-LIST - These verbs are used to copy and edit saved select-lists. COPY-LIST will copy a saved select-list to the terminal, to another select-list, or to a file as a new item. EDIT list will let you edit a saved select-list.

T-DUMP, S-DUMP, and T-LOAD - The T-DUMP verb allows the user to selectively dump his dictionaries and data files to the tape or cartridge disk. The S-DUMP verb performs the same as the T-DUMP verb, but sorts the data before dumping. The T-LOAD verb allows the selective reloading of data or dictionary items that have been previously dumped to tape or disk cartridge using T-DUMP or S-DUMP. Data on T-DUMPed or S-DUMPed media may also be listed by using the LIST, LIST-ITEM, or LIST-LABEL verbs with the TAPE modifier.

LIST-ITEM and SORT-ITEM - The LIST-ITEM and SORT-ITEM verbs facilitate the dumping of the contents of selected items to the user's terminal or to the line printer. The items will be dumped in EDITOR format, with line numbers to the left. This kind of dump differs from a COPY dump in that SORT-ITEM and LIST-ITEM are ACCESS verbs, whereas COPY is a TCL-II verb. This means that SORT-ITEM and LIST-ITEM sentences may contain selection criteria, headings, and footings, none of which are available to the COPY processor.

ISTAT and HASH-TEST - HASH-TEST and ISTAT provide useful file management information. These verbs give a file hashing histogram and file utilization statistics; the ISTAT verb provides information for an existing file and the HASH-TEST verb uses a test modulo to provide information about a file typically prior to the reallocation of the extents of the file.

CHECK-SUM - This verb generates a checksum for items in a file so that you can determine whether data in a file has been changed.

Examples of the use of ACCESS verbs:

```
    >LIST ACCOUNT NAME CURR-BALNC WITH CURR-BALNC  [CR]

    >SORT ACCOUNT >"10000" WITH CURR-BALNC  [CR]

    >LIST-LABEL ACCOUNT NAME ADDRESS (N)  [CR]

    >SORT-LABEL ACCOUNT NAME ADDRESS BY BILL-RATE LPTR  [CR]

    >SUM FILE4 QUAN  [CR]

    >SELECT DICT MD WITH D/CODE "D"  [CR]

    >SELECT ACCOUNT WITH BILL-RATE = "10.03"  [CR]

    >ISTAT ACCOUNT (P)  [CR]

    >SORT-ITEM BP # "*]" AND # "$]" (PF)  [CR]
```

## 1.4 <u>THE LIST VERB</u>

LIST is used to generate a formatted output of selected items and attributes from a specified file. An ACCESS sentence using the LIST verb has the following general form:

```
    LIST {DICT} file-name {item-list} {selection-criteria}
         {output-specifications {print-limiters}}
         {modifiers} {(options)}
```

The optional DICT modifier specifies that the dictionary section of the file, as opposed to the data section, is to be listed. The file-name is the name of the file, and must be present in the user's master dictionary (MD). The optional item-list enumerates specific items to be listed. The optional selection-criteria will limit the items to be listed to those meeting some user-defined set of specifications. A detailed discussion of the selection processor is given in Section 1.18. The output-specifications, if present, indicate to the LIST processor just which attributes (fields) of the selected items (records) are to be listed.

The selected items will be listed on the user's terminal, unless the LPTR modifier or 'P' option was used. If an item-list is used, items will be listed in the same order as the item-ids appear in the item-list. If no item-list is specified, all items in the file will be listed, and they will appear in order of the group they hash into, and within groups by order of when they were added to the file.

The LIST verb will provide information on any or all items in a file. It can be particularly useful if the user only wishes to see information on a small number of items.

Consider the following example:

```
    >LIST ACCOUNT "35000""35050" NAME ADDRESS  [CR]
```

This LIST sentence specifies that the attributes (fields) named NAME and ADDRESS in the items (records) having item-ids (keys) "35000" and "35050" in the file ACCOUNT are to be listed.

To query the file for items meeting a set of specifications, selection criteria are used. For example:

```
    >LIST ACCOUNT WITH NAME "J J JOHNSON"  [CR]
```

The LIST sentence specifies that all items whose NAME is "J J JOHNSON" in the file named ACCOUNT are to be displayed, along with their item-ids. Thus, the entire file will be queried to discover which items meet these user-defined specifications.

Further examples of the LIST verb:

```
    >LIST ACCOUNT WITH BILL-RATE "30" NAME ADDRESS BILL-RATE  [CR]

    PAGE 1                                          11:08:37  13 SEP 1982

    ACCOUNT... NAME.............       ADDRESS............. BILL-RATE

    11115      D R MASTERS             100 AVOCADO          30
    11085      A B SEGUR               101 BAY STREET       30
    11040      B G MCCARTHY            113 BEGONIA          30
    11050      J R MARSHECK            125 BEGONIA          30
    11020      J T O'BRIEN             124 ANCHOK PL        30
    11095      J B STEINER             124 AVOCADO          30
    11110      D L WEISBROD            106 AVOCADO          30
    11015      L K HARMAN              118 ANCHOR PL        30
    11105      C G GREEN               112 AVOCADO          30
    11090      J W JENKINS             130 AVOCADO          30
    23030      L J DEVOS               201 CARNATION        30

    11 ITEMS LISTED.

    >LIST ACCOUNT > "23080" AND <= "23095" NAME ADDRESS  [cs]_  [CR]
    :START-DATE CURR-BALNC DEPOSIT  [CR]

    PAGE 1                                             11:19:58  13 JUL 1982

    ACCOUNT    : 23095
    NAME         W E ZUMSTEIN
    ADDRESS      224 BEGONIA
    START-DATE   02 JAN 1980
    CURR-BALNC   $23.50
    DEPOSIT      11.00

    ACCOUNT    : 23081
    NAME         J W YOUNG
    ADDRESS      207 COVE STREET
    START-DATE   27 MAR 1975
    CURR-BALNC   $89.32
    DEPOSIT      10.00

    ACCOUNT    : 23090
    NAME         W J HIRSCHFIELD
    ADDRESS      230 BEGONIA
    START-DATE   01 JAN 1968
    CURR-BALNC   $20.45
    DEPOSIT      10.00

    3 ITEMS LISTED.
```

## 1.5 THE SORT VERB

SORT is used to generate a sorted and formatted output of selected items and attributes from a specified file. The general form of the SORT verb is:

```
    SORT {DICT} file-name {item-list} {selection-criteria}
         {sort-keys} {output-specifications {print-limiters}}
         {modifiers} {(options)}
```

The output produced by a SORT operation is identical to the output produced by a LIST operation except that the SORT orders the output in a user-specified order. Sort-keys are specified by the following modifiers:

```
    BY attribute-name
    BY-DSND attribute-name
    BY-EXP attribute-name
    BY-EXP-DSND attribute-name
```

The attribute name immediately following one of these modifiers in the SORT sentence will be used as a sort-key. The "-DSND" suffix to a modifier specifies descending order; the default is ascending order. The "-EXP" suffix specifies that the attribute specified by the attribute name has multiple values, so that multiple sort-keys may be generated for each item.

If no sort-keys are specified, the item-ids will be used as sort-keys, and sorting will be in ascending order. A descending sort on item-ids may be produced by sorting on an attribute name that is synonymous with the item-id (has a 0 in attribute 2). Multiple sort-keys may be intermixed at will, with the leftmost sort-key being the most significant. That is, the items will first be sorted by the sort-key which appears first in the ACCESS sentence, then by the next sort-key on the right, and so on.

If the sort-key attribute(s) is left-justified, (has an L in attribute 9), sequencing of a SORT operation is accomplished by comparing the ASCII character representations of the sort-key attribute(s) specified from left to right.

If the sort-key attribute name is right-justified, (has an R in attribute 9), then a numeric comparison is performed; if the data is alphanumeric, numeric portions of the keys are compared numerically, and non-numeric portions are compared left to right. (Note the difference in this comparison technique from that used in selection-criteria.) Consider the following example:

```
    >SORT INV BY QUAN BY PRICE  [CR]
```

This sentence specifies that all items in the file named INV are to be sorted, first by the attribute named QUAN and within that by the attribute named PRICE.

Additional examples are shown below:

```
    >SORT ACCOUNT GE "23000" AND LE "23020" NAME START-DATE  [CR]

    PAGE 1                                        14:11:02  22 NOV 1982

    ACCOUNT... NAME...............    START-DATE.

    23000      H T LEE                01 JAN 1978
    23005      W B THOMPSON           29 DEC 1979
    23010      W E MCCOY              01 JAN 1978
    23015      R M COOPER             01 JAN 1978
    23020      S L UNDERLEIDER        23 APR 1982

    5 ITEMS LISTED.

    >SORT ACCOUNT WITH CURR-BALNC > "95000" NAME CURR-BALNC  [cs]_  [CR]
    :BY-DSND CURR-BALNC  HDR-SUPP  [CR]

    ACCOUNT... NAME...............    CURR-BALNC.

    11055      W H KOONS              $958,343.75
    35080      G A BUCKLES            $447,765.48
    11020      J T O'BRIEN            $306,755.54
    23040      P B SCIPMA             $123,423.22
    23045      P F KUGEL              $ 99,422.34

    >SORT ACCOUNT > "25070" NAME DEPOSIT BILL-RATE  [cs]_  [CR]
    :BY DEPOSIT BY-DSND BILL-RATE  [CR]

    PAGE 1                                        14:15:47  25 OCT 1982

    ACCOUNT... NAME..............     DEPOSIT BILL-RATE

    35090      D U WILDE               3.17       10.03
    35100      R W FORSTROM            8.00       10.03
    35110      H E DAPLOWITZ          10.00       10.03
    35080      G A BUCKLES            10.00         .35
    35095      A W FEVERSTEIN         10.00         .35
    35105      S J FRYCKI             10.00         .35
    35075      J L CUNNINGHAM         10.00         .30
    35085      J F SITAR              12.00         .02

    8 ITEMS LISTED.
```

In generating the values used in sort-key comparison, correlatives in the attribute definition are processed, but conversion specifications are not. Note that conversion codes (MR, ML, and D) do not affect the sort results and should not be used as correlatives in the attribute names which make up sort-keys in order to save processing time.

## 1.5.1 EXPLODING SORT ON MULTIVALUED FIELDS; BY-EXP AND BY-EXP-DSND

The Exploding Sort on multivalues allows system processors to access individual values in a multivalued item and to sort that item according to any specified value or values.

This capability "explodes" the item into effectively multiple items, with the explosion being controlled by the multivalued attribute(s) in the data.

The exploding sort modifiers may be used in SORT or SSELECT sentences. If SSELECT is used with the exploding modifiers, the select-list that is generated will have not only the item-id, but the value-number of the value within its multivalued set stored in the string. This value-number is accessible to BASIC programs via the READNEXT var,vmc form of the BASIC READNEXT statement.

Explosion is specified by using the BY-EXP modifier instead of the BY modifier (or the BY-EXP-DSND instead of the BY-DSND). The attribute that follows the BY-EXP may be multivalued; there will be as many pseudo-items created as there are values in the attribute. If multiple BY-EXPs are specified, the attribute with the maximum number of multiple values will be used to create the items; the fields where there is no data for a value will be treated as null.

The general form of the BY-EXP is:

```
    BY-EXP attribute-name {"explosion-limiter"}
```

where the explosion-limiter is of the same form as the print-limiter (see Section 1.20, PRINT LIMITERS), and serves to limit the explosion to only those multivalues that pass the limiting condition.

A single-valued attribute that is specified with an exploding sort modifier will have its single value repeated in each occurrence of the item.

Sorts may be performed in ascending order using the "BY-EXP" modifier or in descending order, using the "BY-EXP-DSND" modifier. The following example demonstrates how an Exploding Sort may be used to sort items according to one value.

A publisher's mailing-list file is comprised of items which contain the names, addresses, special type code and subscription dates of its customers. The item-id of each item is a last name concatenated with a first initial, and the item contains the information for customers whose last name and first initial are the same as the item-id.

The attributes in the item are ordered as follows:

| | |
|---|---|
| Attr. 1 | First Name and Middle Initial |
| Attr. 2 | First Line of Address |
| Attr. 3 | Second Line of Address |
| Attr. 4 | Zipcode |
| Attr. 5 | Special Type Code |
| Attr. 6 | Subscription Date |

Each attribute contains a number of values; one value per customer. For the sake of demonstration, we will assume the file 'MASTER' contains only the following two items:

```
Item-Id:  SMITH*J
Attr. 1   JOHN T]JIM W]JANET]JOSEPH K
Attr. 2   755 PARK]BOX 301] 77 SUNSET]405 NASTER
Attr. 3   N.Y. NY]PLAINS GA]MIAMI FL]IRVINE CA
Attr. 4   10022]31780]33147]92714
Attr. 4   4D]7D]9E]3E
Attr. 6   6/66]8/72]4/76]11/82

Item-Id :  JONES*T
Attr. 1   TOM F]TERRY]TEDDY]TIM
Attr. 2   1 APPLE]56 FIRST]45 HOLLY]112 ELM
Attr. 3   AKRON OH]MODESTO CA]TAMPA FL]JACKSON MS
Attr. 4   44306]95351]33624]39211
Attr. 5   6D]2D]7D]5D
Attr. 6   4/75]3/76]11/81]4/73
```

A listing of the mailing list sorted by zip code can be obtained with the following ACCESS statement:

```
SORT MASTER ID-SUPP BY-EXP ZIP FNAME LNAME ADR1 ADR2 ZIP TCODE DATE
```

The listing is sorted in the order of the zip codes of each value as shown below:

```
FIRST NAME   LAST NAME   ADDRESS      STATE        ZIP      CODE DATE

JOHN T       SMITH       755 PARK     N.Y. NY      10022    4D   6/66
JIM W        SMITH       BOX 310Y     PLAINS GA    31780    7D   8/72
JANET        SMITH       77 SUNSET    MIAMI FL     33147    9E   4/76
TEDDY R      JONES       45 HOLLY     TAMPA FL     33624    7D   11/81
TIM          JONES       112 ELM      JACKSON MS   39211    5D   4/73
TOM F        JONES       1 APPLE      AKRON OH     44306    6D   4/75
JOSEPH K     SMITH       405 NASTER   IRVINE CA    92714    3E   11/82
TERRY        JONES       56 FIRST     MODESTO CA   95351    2D   3/76
```

If the preceding sort had employed the BY-EXP-DSND modifier, then the data would have sorted in the reverse sequence, that is, from the highest zip code to the lowest.

The exploding-sort modifiers may also be used with explosion limiters. For example, the ACCESS statement:

```
SORT MASTER BY-EXP ZIP > "39999" ID-SUPP FNAME LNAME ADR1 ADR2 ZIP CODE DATE
```

yields only those values whose corresponding zip code is greater than 39999:

```
FIRST NAME  LAST NAME   ADDRESS      STATE        ZIP    CODE  DATE

TOM F       JONES       1 APPLE      AKRON OH     44306  6D    4/75
JOSEPH K    SMITH       405 NASTER   IRVINE, CA   92714  3E    11/82
TERRY       JONES       56 FIRST     MODESTO CA   95351  2D    3/76
```

## 1.6 THE LIST-LABEL AND SORT-LABEL VERBS

LIST-LABEL and SORT-LABEL are ACCESS verbs that will print mailing labels or produce other special purpose listings.

The LIST-LABEL and SORT-LABEL verbs are almost identical in function to the LIST and SORT verbs, the only difference is that the LIST-LABEL and SORT-LABEL listings may have more than one item on each line. Thus, the data associated with each item can be grouped into blocks, and several of these blocks may be placed across each page of the listing.

LIST-LABEL and SORT-LABEL sentences have exactly the same format as LIST and SORT sentences respectively, except that an additional set of parameters is requested from the user after the ACCESS sentence has been entered. The ACCESS sentence is entered in the same manner as any other ACCESS sentence, either through the user´s terminal or in a PROC. The additional parameters are then entered either through the user´s terminal or in a PROC and passed through the Secondary Output Buffer. The system prompts for the additional parameters on the terminal with a question mark (?) prompt until a null line of data ([CR]) is entered.

The formats for the LIST-LABEL and SORT-LABEL sentences are:

```
    LIST-LABEL {DICT} file-name {item-list} {selection-criteria}
               {output-specifications {print-limiters}}
               {modifiers} {(options)}

                             ...and...

    SORT-LABEL {DICT} file-name {item-list} {selection-criteria}
               {sort-keys} {output-specifications {print-limiters}}
               {modifiers} {(options)}
```

Refer to LIST and SORT verbs for information on all parameters shown above.

The first additional line of parameters, which must be input after any ACCESS sentence containing a LIST-LABEL or SORT-LABEL verb, has the following format:

```
    ?count,rows,skip,indent,size,space{,C}
```

These parameters determine the arrangement of attribute data into lines of labels. They are entered in the following order, with values separated by commas:

| Parameter | Meaning |
|---|---|
| count | The number of items (labels) across each page |
| rows | The number of lines printed for each label (height of each label, in rows) |
| skip | The number of lines to skip between labels (horizontal spacing between labels, in rows) |
| indent | The number of spaces to indent the data from the left margin |
| size | The maximum width allowable for the data associated with each attribute name (width of each label, in columns) |
| space | The number of spaces between labels (vertical spacing between labels, in columns) |
| C | Optional; if present, specifies that null attributes are not to be printed. If the C is not specified, null values will be printed as all blanks. |

Values must conform to the range:

```
    (count * (size + space) + indent) <= (current page width)
```

where "current page width" is the number defined for the current output device (terminal or line-printer) by the TERM verb. Otherwise, the system will respond with error message 290:

```
    [290] THE RANGE OF THE PARAMETER "parameter" IS NOT ACCEPTABLE.
```

The normal non-columnar list heading (page number, time and date) will print on the top of each page, unless suppressed by the COL-HDR-SUPP modifier or (C) option. If headings are suppressed, pagination and all top-of-forms are suppressed, which produces a continuous forms format without page breaks.

If the parameter "indent" is non-zero, a set of row header data lines will be requested. These requests will immediately follow the first request for the six numeric parameters, and are prompted for with question marks. The parameter "rows" specifies how many row headers will be requested, because one row header will be printed for each row in the labels. When the listing is printed, these headers will appear in the left-hand margin or "indent" area. Null headers may be specified by entering null lines ([CR]) to the header data requests.

An example of an ACCESS sentence containing a SORT-LABEL verb:

```
>SORT-LABEL NAMES WITH LAST.NAME = "[SON" BY LAST.NAME [cs]_  [CR]
:BY FIRST.NAME NAME ADDRESS CITY/STATE ZIP  [CR]

?3,4,1,13,14,5,C
?CUST.NAME.
?ADDRESS...
?CITY/STATE
?ZIP CODE..

PAGE 1                                       11:22:33  25 OCT 1982

CUST.NAME.       AARON AARONSON     ABE AARONSON     CITY OF CARSON
ADDRESS...       1213 N. ELAINE     18236 FOXGLOVE   P.O. BOX 9905
CITY/STATE       DAYTON, OHIO       BOSTON, MASS.    CARSON, CALIF.
ZIP CODE..                45414              02122            90712

CUST.NAME.       JACK JOHNSON       KELLY JOHNSON    LARRY JOHNSON
ADDRESS...       845 AVOCADO        20650 MARCHETA   525 DUNNEGAN
CITY/STATE       PHOENIX, ARIZ.     ALBUQ., N MEX.   BOSTON, MASS.
ZIP CODE..                85019              87105            02158

CUST.NAME.       MABELL JOHNSON     NEDLIE JOHNSON   TOM THOMPSON
ADDRESS...       210 9TH STREET     4333 N. MAIN     1881 MITCHELL
CITY/STATE       SANTA MARIA,CA     SANTA MONICA,C   SAN DIEGO, CA
ZIP CODE..                93454              90402            92126

9 ITEMS LISTED.
```

## 1.7 THE REFORMAT AND SREFORMAT VERBS

REFORMAT and SREFORMAT are equivalent to LIST and SORT, except that the output is directed to another file or to tape or cartridge disk instead of a terminal or line printer.

The REFORMAT and SREFORMAT verbs are almost identical in function to the LIST and SORT verbs, the only difference is that the output is not made into a listing. Instead, the output can be used as data to update items in a file, to change the arrangement of attributes in a file, or to write tape or cartridge disk records.

REFORMAT and SREFORMAT sentences have exactly the same format as LIST and SORT sentences, respectively. However, the user must supply another file name after the ACCESS sentence is provided.

The formats for the REFORMAT and SREFORMAT sentences are:

```
    REFORMAT {DICT} file-name {item-list} {selection-criteria}
             {output-specifications {print-limiters}}
             {modifiers} {(options)}

                              ...and...

    SREFORMAT {DICT} file-name {item-list} {selection-criteria}
              {sort-keys} {output-specifications {print-limiters}}
              {modifiers} {(options)}
```

Note that exploding sort-keys are valid, but control breaks and totals are not.

The REFORMAT or SREFORMAT ACCESS sentence is entered in the same manner as any other ACCESS sentence, either at the terminal or in a PROC.

On execution of REFORMAT or SREFORMAT, the user's terminal will be prompted with:

```
    FILE NAME:
```

At this point, the user enters the name of the destination file where the output of the list processor will be stored, or enters the word TAPE for tape or cartridge disk. The destination file-name is passed through the Secondary Output Buffer when using a PROC. A null response will indicate that the file specified in the ACCESS sentence will be used as a destination file.

NOTE: To REFORMAT a file onto itself, you must specify an item-list or have a select-list present; otherwise, an infinite loop in which items are continuously added to the file may result.

## Reformatting To Another File

When reformatting a file onto another file, the first value defined by the output specifications (i.e., the first column that would appear in a listing) is used as an item-id. The remaining values make up the item. Thus, each line that would have occurred in a listing becomes one item in the destination file. An example of reformatting a file onto another file:

```
>LIST EMPLOYEE NAME SSN (H)  [CR]

EMPLOYEE.......  NAME...............     SSN.........
572-08-3839      GROMAN,M.               572-08-3839
215-54-4351      ROSE, J.                215-54-4351
684-34-1100      CHAPEL,B.               684-34-1100

>REFORMAT EMPLOYEE NAME SSN  [CR]

FILE NAME:NAMES  [CR]

Item-id          Attribute 1

CHAPEL,B.        684-34-1100
GROMAN,M.        572-08-3839
ROSE,J.          215-54-4351
```

## Reformatting to Tape or Cartridge Disk

When reformatting a file to tape or cartridge disk, the values are concatenated together, and either truncated or padded at the end with nulls (hex ´00´s) to obtain a tape block the same length as the current tape block size, as specified by the T-ATT verb. (When using cartridge disk, the block size is always 1024 bytes.)

One tape or cartridge disk record will be written for each line that would have appeared in the listing. A label, which contains the file name, tape block size (in hex), time and date, is written on the tape or cartridge disk first, unless the HDR-SUPP or COL-HDR-SUPP modifiers (H or C options) are specified. Two End-of-File marks (EOF´s) terminate the tape or cartridge disk. Item-ids will be printed as items are dumped, unless the (I) (ID-SUPPress) option is used.

An example of reformatting a file to tape:

```
    >SORT VENDORS NAME PHONE ADDRESS HDR-SUPP  [CR]

    VENDORS...  NAME................. PHONE...   ADDRESS....................

      12345 JACKSON ENTERPRISES       523-3688   P.O. BOX 322 JACKSON, MISS
      12888 ACME BOLT CO.             444-8819   17911 SKY PARK CIR IRVINE, CA.
      12900 SPACE PRODUCTS, INC.      444-3122   99 E. 15TH ST. MOBILE, AL.

    >T-ATT (80)  [CR]       (Specifies 80-byte tape records.)
    TAPE ATTACHED

    >SREFORMAT VENDORS NAME PHONE ADDRESS  [CR]

    FILE NAME:TAPE  [CR]

        1  12345
        2  12888
        3  12900
    3 ITEMS DUMPED.

    >T-REW  [CR]

    >T-READ  [CR]

    L 0050  12:18:15  28 JUNE 1982  VENDORS

    RECORD = 1

     1 JACKSON ENTERPRISES 523-3888P.O. BOX 322 JACKSON,
    51 MISS.    _....................

    RECORD = 2

     1 ACME BOLT CO.       444-881917911 SKY PARK CIR IRV
    51 INE, CA._....................

    RECORD = 3

     1 SPACE PRODUCTS, INC.444-312299 E. 15TH ST. MOBILE,
    51 AL.      _....................

    [94]  END OF FILE
```

## 1.8 THE COUNT VERB

COUNT is an ACCESS verb which counts the number of items in a file. The items may be limited to those that meet the conditions of an item-list, selection-criteria, or a combination of these.

An ACCESS sentence using the COUNT verb may contain any item-lists or selection-criteria valid for a LIST or SORT sentence, but no sort-keys, break-ons, totals or output specifications will be processed. This gives the COUNT sentence the following general form:

```
COUNT {DICT} file-name {item-list} {selection-criteria}
     {(options)}
```

The COUNT verb will generate error message 407:

```
n ITEMS COUNTED.
```

where "n" is the number of items meeting the specifications set down by the item-list and selection-criteria, if present. If neither item-list nor selection-criteria are specified, the number (n) returned will be the number of items in the specified file. The maximum number of items which can be counted is 2,147,483,647 (Hex "7FFFFFFF").

Consider the following example:

```
COUNT AFILE > "553" WITH A3 = "ABC"
```

This sentence counts the items in the file named AFILE which have item-ids greater than 533 and also have a value of ABC for the attribute named A3.

Other examples of COUNT usage:

```
>COUNT TEST  [CR]
10 ITEMS COUNTED.

>COUNT ACCOUNT WITH BILL-RATE "30"  [CR]
11 ITEMS COUNTED.

>COUNT ACCOUNT GE "11115" WITH CURR-BALNC AND WITH BILL-RATE "30"  [CR]
2 ITEMS COUNTED.
```

## 1.9 THE SUM AND STAT VERBS

SUM generates the total of all the values of one attribute-name for a selected set of items in a file. STAT works like SUM, but STAT also provides a count of the number of items selected and an average value for the attribute-name. If only the file-name is given, SUM and STAT will give the total byte count for the file.

An ACCESS sentence using the SUM or STAT verbs has the same general form as a sentence using the COUNT verb:

```
    SUM {DICT} file-name {item-list} {selection-criteria}
        {(options)}

   STAT {DICT} file-name {item-list} {selection-criteria}
        {(options)}
```

The output produced by a SUM operation has the following general form:

```
    TOTAL OF aaaa = xxxx
```

where aaaa is the header for the attribute-name, (the text from line 3 of the attribute-name dictionary, or the item-id if line 3 is null), and xxxx is the computed total for the attribute-name. Examples of the use of the SUM verb:

```
    >SUM ACCOUNT CURR-BALNC  [CR]

    TOTAL OF CURR-BALNC IS : $2,405,118.10

    >SUM ACCOUNT CURR-BALNC WITH CURR-BALNC > "100000"  [CR]

    TOTAL OF CURR-BALNC IS : 1,836,287.99

    >SUM ACCOUNT > "35055" CURR-BALNC  [CR]

    TOTAL OF CURR-BALNC IS : $605,916.48

    >SUM DICT ACCOUNT V/MAX  [CR]

    TOTAL OF V/MAX IS : 2887

    >SUM ACCOUNT DEPOSIT WITH CURR-BALNC LT "50000"  [CR]

    TOTAL OF DEPOSIT IS : 542.17

    >SUM ACCOUNT DEPOSIT WITH CURR-BALNC <"50000"& WITH NO SEWER-ASMT  [CR]

    TOTAL OF DEPOSIT IS : 460.00
```

The output of the STAT verb has the following general form:

```
    STATISTICS OF aaaa :
    TOTAL = xxxx AVERAGE = yyyy COUNT = zzzz
```

where aaaa is the header for the attribute name, xxxx is the generated sum of all the values for the attribute-name (total), zzzz is the number of items selected (count), and yyyy is the average value for the attribute-name (total divided by count). Examples of the use of the STAT verb:

```
    >STAT ACCOUNT TRASH-CHGS  [CR]

    STATISTICS OF TRASH-CHGS :
    TOTAL = 504.94 AVERAGE = 7.4255 COUNT = 68

    >STAT ACCOUNT CURR-BALNC WITH TRASH-CHGS GE "7.4255"  [CR]

    STATISTICS OF CURR-BALNC :
    TOTAL = 1,199,466.82 AVERAGE = $57,117.4676 COUNT = 21

    >STAT ACCOUNT "11065" "23055" "35050" "35085" BILL-RATE  [CR]

    STATISTICS OF BILL-RATE :
    TOTAL = 57 AVERAGE = 14.25 COUNT = 4

    >STAT ACCOUNT DEPOSIT WITH NO CURR-BALNC  [CR]

    STATISTICS OF DEPOSIT :
    TOTAL = 39.00 AVERAGE = 7.8000 COUNT = 5
```

NOTE: Correlatives are processed in determining totals for the SUM and STAT verbs, but conversions are not processed. Any conversions present will be applied just before the total is printed. (See Section 2.4, CONVERSION AND CORRELATIVE CODES.)

## 1.10 THE CHECK-SUM VERB

This verb generates a checksum for file items, which lets you see if data in file has been changed. The general form of the CHECK-SUM verb is as follows:

```
    CHECK-SUM {DICT}file-name{item-list}{attribute}
              {selection-criteria}
```

where:

| | |
|---|---|
| DICT | dictionary portion of file is checked. |
| file-name | name of file being checked. |
| item-list | checks only those items specified. |
| attribute | checks only specified attribute. |
| selection-criteria | checks only those items which satisfy specified criteria. |

The CHECK-SUM command generates a checksum for file items, thus providing a means to determine if data in a file has been changed. A checksum is the arithmetic total, disregarding overflow, of all bytes in the selected items.

A checksum is generated for items in the specified file, or subset of items if the optional "item-list" and/or "selection-criteria" appear. Furthermore, the checksum may be calculated for one specified attribute. The dictionary portion is checksummed if {DICT} appears. If no attribute is specified, the first default attribute will be used. If there is no default attribute, or if the AMC is 9999, the entire item will be included. The checksum will include the binary value of each character times a positional value.

This yields a checksum which has a high probability of being unique for a given character string. A message is output giving checksum statistics in the following form:

```
    BYTE STATISTICS FOR file-name (or attribute name):
    TOTAL = t AVERAGE = a ITEMS = i CKSUM = c BITS = b
    t     is the total number of bytes in the attribute (or item) included
    a     is the average number of bytes
    i     is the number of items
    c     is the checksum
    b     is the bit count
```

The attribute mark trailing the specified attribute (or item) will be included in the statistics. To use checksums, the user should issue CHECK-SUM commands for all files or portions of files to be verified and keep the output statistics. Subsequently, the CHECK-SUM commands can be reissued to verify that the checksum statistics have not changed. The checksum must be recalculated whenever the user updates the file.

## 1.11 THE SELECT, SSELECT AND QSELECT VERBS

SELECT allows you to select a set of items or attribute values from a file and generate a select-list of the selected item-ids or values. The SSELECT verb combines the SORT capability with the SELECT capability. The QSELECT verb is used to quickly generate a select-list from attribute(s) within item(s) in a file.

A sentence constructed with the SELECT or SSELECT verb will have the same general form as a sentence using a LIST or SORT verb, respectively:

```
    SELECT {DICT} file-name {item-list} {selection-criteria}
           {output-specifications {print-limiters}}
           {modifiers} {(options)}

                              ...and...

    SSELECT {DICT} file-name {item-list} {selection-criteria}
            {sort-keys} {output-specifications {print-limiters}}
            {modifiers} {(options)}
```

The difference between the SELECT and LIST verbs (or between the SSELECT and SORT verbs) is in the handling of the attribute name output as specified by the output specifications. With the LIST or SORT verbs, the output for each attribute name is directed to the user's terminal, a line printer, a tape or cartridge disk unit, or a SPOOLER hold file. With the SELECT or SSELECT verbs, the atttribute values and/or item-ids in the output specifications are saved in a select-list. This "select-list" is a temporary list which is stored in the user's workspace until the execution of one more verb. (To permanently save select-lists, see the following section.) The next verb executed will use the select-list as its implicit item list.

If a numeric value is used in the options parameter of the SELECT or SSELECT statement, the select will stop after the number of items found is equal to the specified value.

Examples of the use of SELECT and SSELECT:

```
    >SELECT ACCOUNT WITH SEWER-ASMT  [CR]

    11 ITEMS SELECTED.
    >EDIT ACCOUNT [CR]
         .
         .
    >SELECT ACCOUNT > "11045" WITH CURR-BALNC LE "0"  [CR]

    3 ITEMS SELECTED.
    >COPY ACCOUNT (P)
    >SSELECT ACCOUNT WITH BILL RATE "30" BY NAME  [CR]

    11 ITEMS SELECTED.
    >RUN BP TEST  [CR]
         .
         .
    >SSELECT ACCOUNT > "11025" WITH DEPOSIT "10" BY-DNSND ADDRESS  [CR]

    63 ITEMS SELECTED.
    >ED ACCOUNT  [CR]
         .
         .
    >SSELECT ACCOUNT BY BILL-RATE BY-DSND DEPOSIT BY NAME  [CR]

    67 ITEMS SELECTED.
    >COPY ACCOUNT  (T,N)  [CR]
```

The output from a SELECT or SSELECT sentence is the select-list and error message 404:

```
    n ITEMS SELECTED.
```

where "n" is the number of item-ids or values selected and placed in the select-list.

SELECT is analogous to the LIST verb in that there is no sequencing of the items. SSELECT will order the selected item-ids or values according to the sort keys in the ACCESS sentence, exactly like the SORT verb.

The elements of the select-list may be used as item-ids to reference data in any file, not just the file referenced in the SELECT or SSELECT sentence. For instance, if a SSELECT on one file is followed by a LIST operation on a different file, the sorted list of item-ids generated by the SSELECT will be used as an item list in the LIST. The LIST processor will attempt to look up and list the items in the second file with the same item-ids selected in the first file. If the LIST processor cannot find an item, it will append error message 780:

```
    [780]  ITEM "item-id" NOT ON FILE.
```

to the end of the listing for each item not found.

In the special case where the SSELECT verb is used with a BY-EXP or BY-DSND-EXP modifier, the elements of the select-list will be multivalued, with the first value being the select data, and the second value being a three-digit value mark count. In BASIC, the two values are retrieved by use of the READNEXT ID,VMC form of the READNEXT statement.

The general form of the QSELECT statement is:

```
    QSELECT file-name {item-list} {(n)}
```

where the data is to be extracted from the item(s) contained in the file file-name. The item-list consists of item-ids separated by blanks, or an asterisk(*) specifying all items. All data from the items is stored in the select-list, unless optional (n) specification is used; in this case, only data from the n-th attribute of each item is used. Multiple values or sub-values are stored as separate elements in the select-list.

The message "n ITEMS SELECTED" will appear at the conclusion of the selection, just as if a SELECT or SSELECT statement has been executed; the generated select-list may then be saved using SAVE-LIST, or used in an ACCESS statement or BASIC program.

The item-list may be omitted by preceding the QSELECT statement with a SELECT, SSELECT, GET-LIST or another QSELECT statement.

Note the complementary nature of the QSELECT verb (which creates a select-list from an item or items) and the COPY-LIST to a file (which creates an item from a select-list).

Sample usage of QSELECT verb:

```
    >QSELECT INV-TRANS 0123-889 987-0999 111-5690 (2)  [CR]

    3 ITEMS SELECTED.                   A select-list is built from the second
                                        attribute of the three specified items
                                        from the INV-TRANS file.

    >QSELECT SYSPROG-PL COLD-LIST  [CR]

    31 ITEMS SELECTED.                  A select-list is built from all of the
                                        attributes in the item COLD-LIST in the
                                        file SYSPROG-PL.

    >SSELECT INVENTORY WITH QUANTITY > "200" BY QUANTITY  [CR]
                                        Generates a temporary select-list.
    123 ITEMS SELECTED.
    >QSELECT INV-TRANS (2)  [CR]        Generates a new select-list from the
                                        data in attribute 2 of those items in
    123 ITEMS SELECTED.                 the INV-TRANS file previously selected.
    >SAVE-LIST XYZ  [CR]                Saves this new select-list.
```

A select-list may be permanently saved by use of the SAVE-LIST verb, or it may be padded as a parameter to another ACCESS operation a BASIC program, or a RUNOFF operation.

The following notes apply to other processes on the system immediately after execution of a SELECT or SSELECT sentence.

| | |
|---|---|
| BASIC | The select-list is available to the BASIC program via the READNEXT STATEMENT. The select-list will override the first SELECT statement in a BASIC program. (See BASIC Manual.) |
| ACCESS | The select-list is available to ACCESS verb processors as an implicit item-list, and will override any item lists in the ACCESS sentence. The selection-criteria are still processed. (Refer to Section 1.18, FORMING ITEM-LISTS.) |
| RUNOFF | The select item-ids or values may be inserted into a RUNOFF text by use of the READNEXT command. |
| SAVE-LIST | The select-list may be catalogued into the POINTER-FILE or a user-specified file by use of the SAVE-LIST verb. This saved select-list is now available to any process on any line through the GET-LIST verb. |
| TCL-II | Same as ACCESS. |

It is important to note that only the sentence immediately following the SELECT or SSELECT sentence will have access to the temporary select-list. This means that if the user marks an error entering the next sentence, then the select-list will be lost and must be selected again. This can be avoided by putting both sentences inside a PROC.

Some of the available disk space will be used to store the temporary select-list. This space will be released to the overflow pool (made available again) after the select-list has been processed.

## 1.12 THE SAVE-LIST, GET-LIST AND DELETE-LIST VERBS

The verbs SAVE-LIST, GET-LIST and DELETE-LIST are used to save, retrieve, and delete selected item-lists. These verbs are useful if several processing passes are to be made on the same set of item-ids or attribute values.

The SAVE-LIST verb provides the facility to make a permanent select-list out of a temporary select-list produced by the SELECT, SSELECT and QSELECT verbs. These permanent select-lists are retrievable via the GET-LIST verb, and may be deleted via the DELETE-LIST verb.

The general form of the SAVE-LIST verb is as follows:

```
SAVE-LIST list-name {(F)}
```

The SAVE-LIST verb will catalog the select-list, (save it in overflow frames) and add or update the pointer to the select-list in the POINTER-FILE. You may specify a file other than POINTER-FILE so long as the file exists and has a "DC" in attribute 1 of its dictionary. To do this, use the (F) option and you will be prompted "FILE-NAME:" for the name of your file.

The system will respond with error message 243:

```
[243] 'list-name' LIST SAVED.  n FRAMES(S) USED.
```

where "n" is the number of overflow frames used to store the list. A previously existing catalogued select-list with the same name will be overlaid by the newly catalogued select-list, and any extra disk space thereby released will be returned to the overflow pool.

If a SAVE-LIST command is entered at TCL level, it must immediately follow the SELECT or SSELECT sentence that generated the desired temporary select-list. If the SAVE-LIST sentence is used in a PROC, it must be placed in the PROC Secondary Output Buffer.

The GET-LIST verb looks up a pointer in the POINTER-FILE (or optional file-name if the F option is used) and retrieves the select-list to which it points. The general form of the GET-LIST verb is as follows:

```
GET-LIST list-name {(F)}
```

where list-name specifies which saved list is requested.

A POINTER-FILE item-id will be generated exactly as for a SAVE-LIST verb, and the POINTER-FILE item will be used to find the saved select-list. If the item cannot be found, the system will respond with error message 202:

```
    'list-name' NOT ON FILE.
```

If the item is found, the system will respond with error message 404:

```
    n ITEMS SELECTED.
```

where "n" is the number of item-ids or attribute values in the saved select-list. The select-list is now available for use by various processors. Only one processor will have access to the select-list, just as if a regular select-list had been generated by a SELECT or SSELECT verb.

The DELETE-LIST verb removes a pointer to a select-list from the POINTER-FILE or a user-specified file-name and returns the frames containing the catalogued list to the overflow pool. The F option specifies that the list(s) are in a user-specified file instead of the POINTER-FILE. The user will be prompted to enter the name of this file by the prompt: FILE-NAME: The general form of the DELETE-LIST verb is as follows:

```
    DELETE-LIST list-name(s) {(F)}
```

A POINTER-FILE item-id will be generated, just as if a SAVE-LIST verb had been generated, and the item will be looked up in the POINTER-FILE.

If the item is found, the system will respond with error message 245:

```
    [245] LIST 'list-name' DELETED
```

The pointer item in the POINTER-FILE will be deleted, and the frames used to store the select-list will be released to the overflow pool.

Sample usage of SAVE-LIST, GET-LIST, and DELETE-LIST:

```
>SSELECT ACCOUNT WITH BILL-RATE > ".35" BY NAME  [CR]

24 ITEMS SELECTED.
>SAVE-LIST OVER.35  [CR]

[243] 'OVER.35' LIST SAVED. 1 FRAME(S) USED.

>GET-LIST OVER.35  [CR]

24 ITEMS SELECTED.
>COPY ACCOUNT  [CR]
TO:(NEW-ACCOUNT)  [CR]

24 ITEMS COPIED.

>GET-LIST OVER.35  [CR]
24 ITEMS SELECTED.
>RUN BP BILLING-OVER.35  [CR]

>DELETE-LIST OVER.35  [CR]
[245] LIST 'OVER.35' DELETED.
```

## 1.13 THE COPY-LIST AND EDIT-LIST VERBS

COPY-LIST allows you to copy a saved select-list to the terminal, to another select-list, or to a file as a new item. EDIT-LIST allows you to edit a saved select-list.

The COPY-LIST statement has the following format:

```
    COPY-LIST list-name(s) {(options)}
```

where file-name and list-name are the names specified in the SAVE-LIST. If the F option is not specified, the list is retrieved from the POINTER-FILE.

| Option | Description |
|---|---|
| D | When copying a select-list to another list, the original select-list is deleted after the copy. |
| F | The list(s) are to be found in a user-specified file instead of the POINTER-FILE. The user will be prompted to enter the name of this file by the prompt: FILE NAME: |
| N | When copying a select-list to the terminal, the automatic end-of-page wait is inhibited. |
| O | Overwrites existing item-ids. |
| P | The select-list is copied to the line printer. |
| T | The select-list is copied to the terminal. |
| X | The data is displayed in hexadecimal. |

If the T or P options are NOT specified, the select-list is to be copied to another select-list or to an item; in this case, the system will respond with

```
    TO:
```

The general response to this message is of the form:

```
    {list-name}
or,
    {(file-name {item-id}}
```

If the list-name form is specified, the original select-list is copied to the newly specified list-name. If (file-name is specified, the list is copied to a user-specified file instead of the POINTER-FILE. If a [CR] is entered instead of list-name, the list is copied to the terminal.

The original select-list will be deleted if the D option had been specified. Note that the new select-list will only overwrite an existing select-list with the same list-name if the O option has been specified.

If the form (file-name is specified, the select-list will be converted to standard item format, with each element of the select-list being stored as an attribute. If the resulting item exceeds the maximum size of 32,267 bytes, it will be truncated at that point. If item-id is not specified, the list-name will be used as the item-id.

The EDIT-LIST statement may be used to edit a previously saved select-list; its general form is:

```
    EDIT-LIST list-name(s) {(F)}
```

The system EDITOR will be entered, and each element in the select-list will be treated as a line in the EDITOR. If the F option is not specified, the list will be retrieved from the POINTER-FILE. If F is spcified, the user will be prompted to enter the name of the file where the list is to be found. All normal EDITOR functions, including MERGE, are valid.

Sample usage of COPY-LIST, and EDIT-LIST verbs:

```
    >COPY-LIST ABC  [CR]
    TO: DEF  [CR]                     Copies select-list ABC to another
                                      list called DEF.
    'DEF' LIST SAVED. 7 FRAMES USED.

    >COPY-LIST ALIST (T)  [CR]        Copies list ALIST to the terminal.

    001 0123-889
    002 987-0999                      Data from the select-list elements.
    003 111-5690

    >COPY-LIST ALIST  [CR]            Copies list ALIST to item "AL"
    TO: (TEST-FILE  AL  [CR]          in the TEST-FILE.

    >EDIT-LIST ALIST  [CR]            Edits the select-list ALIST in the
    TOP                               POINTER-FILE. EDITOR message; top of
    .                                 item.
```

## 1.14 THE ISTAT AND HASH-TEST VERBS

ISTAT and HASH-TEST are ACCESS verbs which provide file utilization information.

A sentence using the ISTAT verb has the following general form:

```
    ISTAT {DICT} file-name {item-list} {selection-criteria}
          {modifiers} {(options)}
```

where ´file-name´ is the name of the file for which the user desires to see the hashing statistics. The ISTAT verb provides a file hashing histogram (bar graph) for the selected items in the selected file, as well as the item count, the total number of bytes in all the items tested, the average number of bytes per item, the average number of items per group and standard deviation, and the average number of bytes per group.

The average number of bytes per group is an excellent indicator as to how well the file space is being utilized. If each group is less than 500 bytes, no additional frames of overflow are needed for the file.

However, if each group is 501 bytes, an additional frame will be linked to each group in the file in order to contain the extra byte, and the remaining 499 bytes in the linked frame will be wasted. (See the second example of the HASH TEST verb.)

If S is entered in the options parameter, the histogram output will be suppressed.

Sample usage of the ISTAT verb:

```
    >ISTAT TEST-FILE  [CR]

    FILE= TEST-FILE MODULO= 29 SEPAR= 1                  16:11:30  27 DEC 1982
    FRAMES BYTES ITMS
         2 05644 023 *>>>>>>>>>>>>>>>>>>>>>>>
         1 05645 022 *>>>>>>>>>>>>>>>>>>>>>>
         1 05646 018 *>>>>>>>>>>>>>>>>>>
            .
            .
            .
        1 05671 017 *>>>>>>>>>>>>>>>>
        3 05672 025 *>>>>>>>>>>>>>>>>>>>>>>>>
        1 05763 020 *>>>>>>>>>>>>>>>>>>

    ITEM COUNT=       365, BYTE COUNT=    15993, AVG. BYTES/ITEM=     43.8
    AVG.ITEMS/GROUP= 12.5, STD. DEVIATION=  3.7, AVG. BYTES/GROUP=    551.4.

    >ISTAT TEST-FILE > "100" AND < "400"  [CR]

    FILE= TEST-FILE MODULO= 29 SEPAR= 1                  16:11:30  27 DEC 1982
    FRAMES BYTES ITMS
         1 05644 003 *>>>
         1 05645 001 *>
         1 05646 000 *
            .
            .
            .
         1 05671 002 *>>
         1 05672 004 *>>>>

    ITEM COUNT=        15, BYTE COUNT=      512, AVG. BYTES/ITEM=     46.3
    AVG.ITEMS/GROUP=  2.6, STD. DEVIATION=  2.3, AVG. BYTES/GROUP=     93.3.
```

The HASH-TEST verb can be used to determine the best modulo for a given file. It shows how any or all of the items in the file would hash into groups for any given modulo, as well as for the other figures and averages provided by the ISTAT verb. The HASH-TEST verb has the following general form:

```
    HASH-TEST {DICT} file-name {item-list} {selection-criteria}
              {modifiers} {(options)}
```

where ´file-name´ is the name of the file to be tested at a new modulo. The optional item-list specifies which particular items are to be tested. If the item list is not specified, all items in the file will be tested. After the HASH-TEST sentence has been entered, the system will prompt for an additional parameter. This parameter, the modulo to be tested, must be entered by the user either from the terminal, or by use of the Secondary Output Buffer in a PROC. The user´s terminal will be prompted with:

```
    TEST MODULO:
```

to which the user enters the modulo (number of groups) with which he wishes to HASH-TEST the specified file. The HASH-TEST verb will then generate a histogram (bar graph) showing the number of items which would hash into each group if the file had the test modulo and a separation of 1. It will also show the statistics generated by the ISTAT verb.

Sample usage of the HASH-TEST verb:

```
>HASH-TEST JUNK-1  [CR]
TEST MODULO:3  [CR]

FILE= JUNK-1 MODULO= 3 SEPAR= 1                       12:33:13  22 FEB 1982
FRAMES BYTES ITMS
     1 00400 001 *>>>
     1 00455 001 *>>>
     1 00345 001 *>>>

ITEM COUNT=          9, BYTE COUNT=     1200, AVG. BYTES/ITEM=     133.3
AVG.ITEMS/GROUP=  3.0, STD. DEVIATION=   .0, AVG. BYTES/GROUP=    400.0.
```

(This is perfect hashing and perfect file utilization.)

```
>HASH-TEST JUNK-2  [CR]
TEST MODULO:3  [CR]

FILE= JUNK-2 MODULO= 3 SEPAR= 1                       12:33:13  22 FEB 1982
FRAMES BYTES ITMS
     2 00600 001 *>>>
     2 00655 001 *>>>
     2 00845 001 *>>>

ITEM COUNT=          9, BYTE COUNT=     2100, AVG. BYTES/ITEM=     233.3
AVG.ITEMS/GROUP=  3.0, STD. DEVIATION=   .0, AVG. BYTES/GROUP=    700.0.
```

(Note that there are now 2 frames in each group.)

## 1.15 THE T-DUMP, S-DUMP AND T-LOAD VERBS, AND THE TAPE MODIFIER

T-DUMP is an ACCESS verb which dumps a specified file to tape or cartridge disk. S-DUMP is the same as T-DUMP except the items in the file are sorted before they are dumped. T-LOAD is used to restore data from a previously generated T-DUMP tape or cartridge disk. The TAPE modifier may be used to list or otherwise access data from a T-DUMP or S-DUMP tape or cartridge disk. Note that a cartridge disk must be formatted before it is used for the first time.

An ACCESS sentence using the T-DUMP verb may specify selection criteria, but no output-specifications. Such a sentence has the following general form:

```
T-DUMP {DICT} file-name {item-list} {selection-criteria}
       {HEADER "name"} {modifiers} {(options)}
S-DUMP {DICT} file-name {item-list} {selection-criteria}
       {HEADER "name"} {sort-keys} {modifiers} {(options)}
```

The tape or cartridge disk unit must be attached by the user before the T-DUMP or S-DUMP commands are issued. The T-ATT command will also set up the tape block size to be used in the T-DUMP or S-DUMP. (The cartridge disk block size is always 1024 bytes.) See T-ATT command in UTILITIES Manual.

T-DUMP and S-DUMP cause a label to be written, followed by a dump of the selected items. If the optional HEADER "name" is specified, the text is added to the standard heading, which is of the form "{DICT} file-name". If the optional DICT is included, the dictionary section of the file will be dumped, and no File Definition Items (with attribute 1 = "D", "DX", "DY", "DC", "DCX" or "DCY") will be dumped. An EOF (End-of-File) mark is written to the tape or cartridge disk after the dump.

The HDR-SUPP connective or (H) option may be used to suppress the label. The ID-SUPP connective, or the (I) option may be used to suppress the listing of item-ids that are being dumped. Sample usage of the T-DUMP and S-DUMP verbs:

```
>T-DUMP ACCOUNT > "23060" WITH CURR-BALNC ID-SUPP  [CR]

31 ITEMS DUMPED
```

This sentence dumps to tape or cartridge disk all items in the ACCOUNT file which have items with item-ids greater than "23060" as well as values for attribute CURR-BALNC.

```
>T-DUMP TEST-FILE  [CR]

    1 A-002
    2 A-088
    3 C-999
    4 A-560
    5 C-888

5 ITEMS DUMPED  [CR]
```

This sentence dumps the entire TEST-FILE file to tape or cartridge disk.

```
S-DUMP FILES  [CR]
```

This command will sort the file into ascending sequence by item-ids, transfer the file to tape or cartridge disk, list the item-ids on the terminal, and write an EOF at the end of the file.

An ACCESS sentence using the T-LOAD verb may specify selection criteria, but no output-specifications. Such a sentence has the following general form:

```
T-LOAD {DICT} file-name {item-list} {selection-criteria}
       {modifiers} {(options)}
```

T-LOAD causes the label to be read from the tape or cartridge disk; this will also set up the tape block size from the label. If unlabeled tapes, or other than the current PICK release tapes are used, the appropriate tape block size must be set up in a T-ATT statement.

The appropriate items, as restricted by the selection criteria or item-list, will be loaded into the file. To overwrite existing items, the (O) option must be specified.

If the ID-SUPP connective, or the (I) option is used, the item-ids of the items being loaded will not be listed.

The tape or cartridge disk will be positioned at the EOF marker at the conclusion of the load. Sample use of the T-LOAD verb:

```
>T-LOAD TEST-FILE  [CR]
'A-002' EXISTS ON FILE.
    1 A-088
    2 C-999
    3 A-560
    4 C-888

4 ITEMS LOADED.
```

This sentence loads all items from the tape or cartridge disk to the TEST-FILE; one item already existed on file, and was not overwritten.

Sample use of the T-LOAD verb: (Continued)

```
>T-LOAD TEST-F "100" AND < "400" (IO)  [CR]

17 ITEMS LOADED
```

This sentence loads only those items from the T-DUMP tape or cartridge disk that have item-ids in the range 100 through 400, even if they already exist on file, and the item-ids are not listed during loading.

The TAPE modifier may be used to read data from a T-DUMP or S-DUMP tape or cartridge disk. The TAPE modifier may be used in any LIST, LIST-LABEL, LIST-ITEM, SUM, STAT, ISTAT, HASH-TEST or COUNT statement; the data items will be retrieved from the tape or cartridge disk file. Note that a file-name must be specified as usual when using the TAPE modifier; the dictionary of this file is still used as the source for the dictionary definitions.

Sample use of TAPE modifiers:

```
>LIST TEST-FILE TAPE  [CR]
```

This sentence will use the dictionary of the TEST-FILE to generate the default output-specifications. It will then read the T-DUMP tape or cartridge disk and format the data items it finds there in the standard listing format.

```
>LIST ACCOUNT WITH CURR-BALNC > "100.00" CURR-BALNC BILL-RATE TAPE  [CR]
```

This statement will select data items from the T-DUMP tape or cartridge disk with CURR-BALNC greater than 100.00, and list the CURR-BALNC and BILL-RATE fields. CURR-BALNC and BILL-RATE are attribute definition items in the dictionary of the ACCOUNT file.

## 1.16 THE LIST-ITEM AND SORT-ITEM VERBS

The LIST-ITEM and SORT-ITEM verbs combine the action of the COPY verb with the selection criteria and heading/footing capabilities of ACCESS verbs.

An ACCESS sentence using the LIST-ITEM or SORT-ITEM verb has the same general form as a sentence using a LIST or SORT verb, except that no output specifications are used. This gives them the following general forms:

```
    LIST-ITEM {DICT} file-name {item-list} {selection-criteria}
              {modifiers} {(options)}

    SORT-ITEM {DICT} file-name {item-list} {selection-criteria}
              {sort-keys}{modifiers} {(options)}
```

Options available for the LIST-ITEM and SORT-ITEM verbs:

| Option | Meaning |
|---|---|
| N | If output is to terminal, inhibits the wait at the end of each page (NOPAGE). |
| P | Output is to line printer (LPTR). |
| F | Starts a new page (a form-feed) for each item. |
| S | Suppresses line numbers. |

The rules for item lists, selection-criteria, sort-keys, modifiers and options for LIST-ITEM and SORT-ITEM sentences are the same as those for LIST and SORT sentences. No output-specifications are given, because instead of listing attribute data, the entire contents of the data items are printed. The items are copied to the user's terminal or to the line printer or spooler just as the COPY verb would copy them, with three-digit line numbers on the left margin. LIST-ITEM differs from COPY in that LIST-ITEM is an ACCESS verb, while COPY is a TCL-II verb. This means that a sentence using the LIST-ITEM verb can specify selection-criteria, and heading and footing text. Sample usage of the LIST-ITEM and SORT-ITEM verbs:

```
>LIST-ITEM BP # "*]" AND # "$]" (P)  [CR]
```

Selects those items in the file named BP which start with characters other than * or $, and copies them to the printer.

```
>SORT-ITEM MD WITH 1 "PQ" HEADING "PROC: '[10' PAGE'P'" (P,F)  [CR]
```

Copies all the user's PROCs (those items in his MD with a PQ in attribute 1) to the line printer, one PROC per page, using the specified heading, in sorted order by item-id.

## 1.17 USING RELATIONAL OPERATORS AND LOGICAL CONNECTIVES

Relational Operators (defined in Table 1-1) may be used in an item-list to constrain the items eligible for processing (see Section 1.17, Forming Item-Lists) or may be used in selection-criteria to limit items to those whose attribute values meet the specified conditions (see Section 1.18, Forming Selection-Criteria). Relational operators apply to the item-id or value immediately following the operator. The absence of a relational operator implies an equality operator.

To resolve a relational condition, every item-id (or attribute value) is compared to the item-id (or value) specified in the item-list (or selection-criteria) of the ACCESS input sentence.

If the attributes or item-ids are left-justified (type code "L" in the dictionary definition), character pairs (one from the specified item-id or value and one from the item-id or attribute currently being compared) are compared one at a time from leftmost characters to rightmost. If no unequal character pairs are found, then the item-ids or values are considered to be "equal." If an unequal pair of characters are found, the characters are ranked according to their numeric ASCII code equivalents. The item-id or value contributing the higher numeric ASCII code equivalent is considered to be "greater" than the other.

If attributes or item-ids are right-justified, a numeric comparison is attempted first. If either or both of the item-ids (values) are non-numeric, the character pair comparison for left-justified attributes is used.

Logical connectives (defined in Table 1-2) bind together sets of item-ids into item-lists, sets of values into value-lists, and sets of selection-criteria into complex selection-criteria. The AND connective specifies that both connected parts must be true, while the OR connective specifies that either (or both) connected parts must be true. In all cases where neither AND nor OR are specified, OR will be assumed.

An ASCII up-arrow (^) may be used as an 'ignore' character in any value or item-id. All comparisons made against the value or item-id then ignore the characters in the corresponding positions. Thus, an up-arrow matches any character.

A left-bracket ([) is a multiple 'ignore' character, which means that all characters to the left of the value or item-id being compared are ignored. Similarly, a right-bracket (]) is a multiple ignore character for the right of the item-id or value being compared. This means that a left-bracket will match any string occurring on the left of a value, including a null string, and a right-bracket will match any string on the right.

The use of the up-arrow and the brackets is further discussed in Section 1.19.1, Selection-Criteria: String Searching.

Table 1-1. Relational Operators

| Operator | Meaning |
|---|---|
| = or EQ | Equal to |
| > or GT or AFTER | Greater than |
| < or LT or BEFORE | Less than |
| >= or GE | Greater than or equal to |
| <= or LE | Less than or equal to |
| # or NE or NOT or NO | Not equal to or null attribute value |
| | If a relational operator is not given, EQ is assumed |

Table 1-2. Logical Connectives

| Connective | Meaning |
|---|---|
| AND | Both connected parts must be true. |
| OR | Either connected part (or both) must be true. If a logical connective is not given, OR is assumed. |

The following examples of relational and logical connectives provide the user a general view of these operators. Complete ACCESS sentences using these operators are presented throughout the remainder of the manual.

| Example | Explanation |
|---|---|
| = "ABC" OR > "DEF" | Item-list which selects item "ABC" as well as all items with item-ids greater than 'DEF'. |
| WITH A1 ="X" AND WITH A2 ="^Z" | Complex selection criterion which selects all items having a value of "X" for attribute A1 and a value for A2 which consists of any character followed by a Z. |
| WITH NAME = "[SMITH" "MEL]" | Selection criterion which selects all items having a value for attribute NAME which either ends with the letters SMITH or begins with the letters MEL. |
| LT "100" GT "200" | Item-list which selects all items with item-ids either less than "100" or greater than "200". |

## 1.18 FORMING ITEM-LISTS

An item-list specifies those items eligible for consideration by the specified processor (verb). There are two types of item-lists: 1) explicit item-lists, which are part of the input ACCESS sentence and 2) implicit item-lists, which are created by the SELECT, SSELECT, QSELECT, and GET-LIST verbs.

### 1.18.1 EXPLICIT ITEM-LISTS

Explicit item-lists consist of one or more specifically enumerated item-ids, enclosed in double quotes (") or backslashes (\).

An item-list defines those items desired for processing. Absence of an item-list implies all items on the file. A simple item-list consists of any number of specified item-ids surrounded by quotes or backslashes (e.g., \XYZ\ or "100-600""100-500""300-000") or a relational operator followed by a single value in quotes (e.g., <"100" or >="SMITH"). A complex item-list consists of sets of simple item-lists bound together with logical connectives (ANDs and ORs).

An explicit item-list, if present, should come right after the file name in the ACCESS sentence. For example, consider the following ACCESS sentence, in which the complex item-list has been underlined:

```
    >LIST TEST-FILE "ABC""XYZ" OR > "DEF" AND < "GHI"
```

This item-list selects items "ABC" and "XYZ", as well as all items with item-ids both greater than "DEF" and less than "GHI".

Use of the complex item-list causes all items in the file to be accessed for examination, as does absence of an item-list. If a simple item-list is used, only those items in the list will be accessed and processing will be faster.

This means that the ACCESS sentence:

```
    >LIST-ITEM BP "STAR-TREK"  [CR]
```

will cause only one item in the BP file, namely, the one whose item-id is "STAR-TREK", to be accessed. Since the item-id is the retrieval key for the item, the item will be accessed immediately. However, the ACCESS sentence:

```
    >LIST-ITEM BP = "STAR-TREK"  [CR]
```

will cause the entire BP file to be searched, with every item-id in the file being matched against the explicit item-list "STAR-TREK". Note that this can be very time consuming for large files.

The hierarchy (precedence) of the logical connectives in an item-list is AND over OR, and left to right. For example, consider the following item-list:

```
    < "A" OR > "B" AND < "C" OR > "D" AND < "E"
```

This item-list selects all items with item-ids less than "A", or with item-ids greater then "B" but less than "C", or with item-ids greater than "D" but less than "E". Since the AND connective has a higher precedence or binding strength than the OR connective, ANDs will be evaluated before ORs, and the above item-list would be evaluated like the following:

```
    <"A" OR (>"B" AND <"C") OR (>"D" AND <"E")
```

Note that the parentheses "(" and ")" are not part of the ACCESS grammer, but are added in the above illustration for clarity.

Since the OR connective is implied if no connective is used, ORs may be omitted from ACCESS sentences. Therefore, the above item-list could have been specified by:

```
    <"A" > "B" AND < "C" > "D" AND < "E"
```

The item-lists may also specify a string-searching capability; this is discussed in Section 6.18.1, Selection Criteria: String Searching.

Further examples of item-lists are illustrated on the following page. Here the SORT verb is used to select and sequence the item-ids in file TEST. (TEST contains ten items, with item-ids "10" through "19".) The word ONLY, as used in these examples, specifies that only the item-ids are to be listed.

```
>SORT ONLY TEST > "13" AND < "17"  [CR]

PAGE 1                                        15:32:19 20 AUG 1982

TEST........

14
15
16

3 ITEMS LISTED.

>SORT ONLY TEST >= "13" AND <="16" OR >="18" AND <"19"  [CR]

PAGE 1                                        15:33:01 20 AUG 1982

TEST.......

13
14
15
16
18

5 ITEMS LISTED.

>SORT ONLY TEST NOT "13" AND NOT "15" AND NOT "17" AND NOT "19"  [CR]

PAGE 1                                        15:33:31 20 AUG 1982

TEST.......

10
11
12
14
16
18

6 ITEMS LISTED.

>SORT ONLY TEST BEFORE "13" [CR]

PAGE 1                                        15:34:24 20 AUG 1982

TEST.......

10
11
12

3 ITEMS LISTED.
```

LIST PM WITH [illegible] AND [illegible]

## 1.18.2 IMPLICIT ITEM-LISTS

Implicit item-lists are formed by the verbs SELECT, SSELECT, QSELECT, and GET-LIST. The next ACCESS sentence executed after the execution of one of these verbs will use the list of items generated by the first verb.

Execution of a SELECT, SSELECT, QSELECT, or GET-LIST verb will result in the message ´n ITEMS SELECTED´, where "n" is the number of items selected and put into the item-list. In the case of a SELECT or SSELECT, the items put into the item-list will be those satisfying the selection criteria (if any) of the SELECT or SSELECT sentence. The item-list generated by a GET-LIST verb is the same item-list that was saved by the use of a SAVE-LIST verb. The item-list generated by a QSELECT depends on the data stored in the items specified in the QSELECT statement.

It is important to note that the use of an implicit item-list will override any explicit item-list. This means that an ACCESS sentence executed after a SELECT, SSELECT, QSELECT, or GET-LIST will use the implicitly specified list and will ignore any explicit item-list.

Selection criteria specified in the statement will, however, be applied as usual to the items in the implicit item-list.

Other SELECT or SSELECT functions can be used on the implicit list obtained from one SELECT, SSELECT, QSELECT, or GET-LIST statement (see the last example on the following page).

Examples of use of an implicit item-list:

```
    >SSELECT TEST  [CR]

    10 ITEMS SELECTED.

    >SAVE-LIST T  [CR]
    [214] 'T' CATALOGED, 1 FRAME(S) USED.

    >SELECT TEST "11""12"  [CR]

    2 ITEMS SELECTED.
    >SORT ONLY TEST  [CR]

    PAGE 1                                          15:32:19 20 AUG 1982

    TEST......

    11
    12

    2 ITEMS LISTED.

    >GET-LIST T

    10 ITEMS SELECTED.
    >LIST ONLY TEST <= "13"  [CR]

    PAGE 1                                          15:32:19 20 AUG 1982

    10
    11
    12
    13

    4 ITEMS LISTED.
```

## 1.19 FORMING SELECTION-CRITERIA

Selection-criteria specify a set of conditions which must be met by an item before it is eligible for output. Complex selection-criteria are made up of one or more simple selection-criteria.

The general form of a selection is as follows:

```
WITH    {EVERY}                           ( relational )
    {NO}            attribute-name        (  operator  )  {value-list}
IF      {EACH}
```

Each selection-criterion must begin with the word WITH or IF followed by a single attribute name. (WITH and IF are synonymous.) The attribute name may then be followed by a value-list. The rules for forming value-lists are identical to those for forming item-lists (see Section 1.17, Forming Item-Lists); double quotes must surround the actual values. For example, the following selection-criterion is met by those items which have at least one value for the attribute DESC which is either equal to "ABC" or is both greater than "DEF" and less than "GHI".

```
WITH DESC "ABC" OR > "DEF" AND < "GHI"
```

If a selection-criterion does not include a value-list, then it is true for all those items which have at least one value for the specified attribute name. The selection-criterion may be further modified by using either or both of the modifiers EVERY or NO immediately following the WITH. The modifier EVERY requires that every value for the attribute meet the specified condition (i.e., if the attribute has multivalues, then each value must meet the condition. (The modifier EACH is a synonym for EVERY.) The modifier NO reverses (inverts) the sense of the entire selection-criterion.

Several selection-criteria may be bound together by logical connectives to form a complex selection-criterion. When used in this fashion, the AND connective has a higher precedence than the OR connective. An AND clause is made up of any number of selection-criteria bound by AND connectives. The AND clause is terminated when an OR connective is found in the left to right scan. Note that the absence of an AND connective implies an OR connective. In order for an item to pass the selection-criteria, the conditions specified by any one of the AND clauses must be met.

An example of the logical hierarchy of AND clauses is shown in the complex selection-criterion below which contains two AND clauses.

```
WITH DESC "ABC" AND WITH VALUE "1000" OR WITH DESC "ABC" AND WITH
NO VALUE
```

Further examples of selection-criteria are:

```
>LIST ACCOUNT NAME WITH AVG-USAGE "20" OR "25" AND [cs_]  [CR]
:WITH SEWER-ASMT "150" OR WITH AVG-USAGE "20" OR "30" [cs_]  [CR]
:AND WITH BILL-RATE < "30"  [CR]

PAGE 1                                              17:36:04  20 AUG 1982

ACCOUNT...  NAME..........  AVG-USAGE   SEWER-ASMT   BILL-RATE

23100       G J PACE           30                       10.30
23080       J W YOUNG          20          1.50          8.40
11045       F R DRESCH         30                       10.03

3 ITEMS LISTED.

>COUNT ACCOUNT WITH CURR-BALNC >"100" AND WITH SEWER-ASMT [cs_]  [CR]
:OR BILL-RATE = "30"  [CR]

7 ITEMS COUNTED.

>LIST ACCOUNT TRNS-DATE WITH EVERY TRNS-DATE BEFORE "3/18/70"  [CR]

PAGE 1                                              17:40:57  20 AUG 1982

ACCOUNT...  TRNS-DATE..
35090       17 MAR 1970
            28 FEB 1970
            17 FEB 1970
            30 JAN 1970
            16 JAN 1970
            29 DEC 1969

END OF LIST
```

## 1.19.1 SELECTION-CRITERIA: STRING SEARCHING

Selection-criteria may be used to search an attribute or an item-id for a string of characters, or to choose attribute values (item-ids) that begin or end with a certain character string.

ACCESS has the ability to search an attribute value or item-id for any string of characters. The left-bracket character ([) and the right-bracket character (]) may be used within the double-quotes in a selection-criterion, or complex item-list to specify a match on any string to the left or right of the given string.

A left-bracket indicates that there may be any (or no) characters to the left of the string. A right-bracket indicates that there may be any (or no) characters to the right of the string. Used separately, the left-bracket specifies that the value must end with the character string, while a right-bracket specifies that the value must begin with the character string. If both brackets are used, the character string may appear anywhere in the attribute value. Examples of the use of string searching selection-criteria:

```
 LIST ACCOUNT WITH NAME "[MAN" NAME  [CR]

PAGE 1                                       18:13:37   20 AUG 1982
ACCOUNT...   NAME......

23025       D C BINGAMAN
23055       S M NEWMAN
11015       L K HARMAN

3 ITEMS LISTED.

 LIST ACCOUNT WITH NAME "A A A]" NAME  [CR]

PAGE 1                                       18:14:09    20 AUG 1982
ACCOUNT...  NAME.....

11070       A A ALTHOFF

END OF LIST

 LIST ACCOUNT WITH NAME "[INE]" NAME  [CR]

PAGE 1                                       18:16:56    20 AUG 1982
ACCOUNT...  NAME.....

11095       J B STEINER
35065       L J RUFFINE

2 ITEMS LISTED.
```

The up-arrow (^) indicates a match on any character. An example of the use of the up-arrow is:

```
 LIST ONLY BP = "^STAR-TREK"  [CR]

PAGE 1                                    18:16:56    20 AUG 1982

BP.......

$STAR-TREK
*STAR-TREK

2 ITEMS LISTED.
```

NOTE

> This string searching capability may not be used in a simple item-list, but may be used with a complex item-list. That is, the simple item-list "[JONES" will only select the item-id "[JONES", if such an item-id is present. The complex item-list = "[JONES" will select any items ending in the string "JONES."

An extended advanced discussion on the use of the selection processor is given at the end of the ACCESS section. See Section 3.1, Using the Selection Processor.

## 1.20 FORMING OUTPUT-SPECIFICATIONS

Output-specifications select the attributes to be listed.

All attribute names in an ACCESS sentence which are not part of a selection-criterion (i.e., preceded by the modifier WITH or IF) or are not modified by certain control modifiers are considered as part of the output specification. These attribute names specify the attribute values which are to be printed out as a result of the specified operation.

If no output-specifications are in the sentence, the system will use the default output-specifications. These default specifications are the attribute names contained in line 3 of the item dictionaries of the file being listed. The dictionary item-ids must be the numbers 1, 2, 3, 4... . The system will sequentially search the dictionary for these numeric items (starting with 1) until it comes to an item-id which is not in the dictionary.

Selected attribute definition items (either specified or default) will be displayed in an automatically generated system format. This format will include a heading line displaying the date, time, and page number (unless supressed) at the beginning of each new page. The page size is set through the use of the TERM command.

The LIST, SORT, LIST-LABEL and SORT-LABEL verbs will attempt to format the output into a columnar format with each specified attribute name as a column heading. A columnar output format example:

```
>SORT ACCOUNT WITH CURR-BALNC > "100000" NAME ADDRESS CURR-BALNC  [CR]

PAGE 1                                           09:09:19  20 AUG 1982

ACCOUNT...  NAME........ ADDRESS..............    CURR-BALNC..

11020       J T O'BRIEN  124 ANCHOR PL            $306,755.54
11055       W H KOONS    131 BEGONIA              $958,343.75
23040       P B SCIPMA   213 CARNATION            $123,423.22
35080       G A BUCKLES  307 DOCK WAY             $447,765.48
4 ITEMS LISTED.
```

The column width (number of spaces) reserved for each attribute definition item will be the size specified in line 10 of the attribute dictionary, or the length of the heading in line 3, whichever is greater. If the sum of the column widths (adding one blank separator for each specified attribute name) does not exceed the page width as set by the TERM command, then a columnar format will be generated. In a columnar format, the specified attribute names are displayed in a heading across the top of the page. The values for each of the items are then displayed in their respective columns. If an attribute has multiple values, each multivalue will be listed on a new line. The heading is repeated at the top of each page.

If the requested output exceeds the page width, then the attribute names are listed down the side of the output with their respective values immediately to the right. A noncolumnar format example:

```
>LIST ACCOUNT "35060" NAME ADDRESS CURR-BALNC BILL-RATE AVG-USAGE  [CR]

PAGE 1                                                09:11:53  20 AUG 1982

ACCOUNT           35060
NAME              J A SCHWARTA
ADDRESS           331 DOCK WAY
CURR-BALNC        $33,822.34
BILL-RATE         2
AVG-USAGE         31

END OF LIST
```

A significant difference between the two formats is that for the columnar format all headings are listed only once for each page, whether or not values exist for the columns, while in the noncolumnar format, headings are only displayed for attribute definition items if the item being listed has values for the specified attribute.

### 1.20.1 DEFAULT OUTPUT-SPECIFICATIONS

If no output-specifications appear in a sentence, a set of default attribute definition items are used to determine the output specifications. The ACCESS processor will look up the items with the sequential item-ids 1, 2, 3, 4 ... in the dictionary of the file being listed. The search for items will continue until an item-id which is not in the dictionary is reached.

If the item with item-id "1" is not found, no output specifications will be used.

The attribute definition dictionary items should have an "A" in line one if the attributes they define are to be listed; an "X" may be specified in line one if the attribute name is not to be listed, but the search for other items is to continue.

Note that the AMC's of these special items 1, 2, etc. need not be in sequence, though typically, the item whose item-id is "1" will have an AMC of 1, the item whose item-id is "2" will have an AMC of 2, etc.

An example of the use of the default set of attribute definition items:

```
>LIST SM82  [CR]

PAGE 1                                         17:45:10  17 NOV 1982

SM82.........      FRM CLASS.... SUB-CLASS..  REV DATE   REV CKSM  LINES OBJ

WHERESUBS          121 SYSTEM    UTILITY      25OCT82    82A C5BE    234 1FB
A-CORR1            231 ENGLISH   CONVERSION   12SEP82    82A 8BE2     43 0D8
BRP1               181 BASIC     RUN-TIME     31OCT82    82B FF37    325 1FF
IDATE              091 ENGLISH   CONVERSION   22SEP82    82A B520    224 1B6
   .
   .
>LIST SM82 HDR-SUPP  [CR]

SM82.........      FRM CLASS.... SUB-CLASS..  REV DATE  REV CKSM   LINES OBJ

WHERESUBS          121  SYSTEM   UTILITY      25OCT82   77A C58E     234 1FB
A-CORR1            231  ENGLISH  CONVERSION   12SEP82   77A 8BE2      43 0D8
   .
>LIST SM82  (C)  [CR]

WHERESUBS          121  SYSTEM   UTILITY      25OCT82   77A C5BE     234 1FB
A-CORR1            231  ENGLISH  CONVERSION   12SEP82   77A 8BE2      43 0D8
   .
   .
```

The ID-SUPP modifier and the I option. Item-ids of the data items will appear leftmost on the listing, underneath the file name in the heading, unless the ID-SUPP modifier or 'I' option is used. (The ID-SUPP, or 'I' option is also used to suppress the listing of item-ids in the T-DUMP and T-LOAD process.)

The HDR-SUPP and COL-HDR-SUPP modifiers. The HDR-SUPP modifier (or 'H' option) will suppress the system generated page heading (time and date on the left, page number on the right). (Note that a HEADING modifier will have this same effect.) The COL-HDR-SUPP (or 'C' option) will suppress the column headers (names from line 3 of the dictionary items) and also suppress the system page heading, and the "n ITEMS LISTED" message at the end of the listing.

The ONLY modifier. Use of the modifier ONLY just before the file name in an ACCESS sentence will cause the default set of attribute definition items to be ignored. An example of the use of ONLY modifier:

```
>LIST ONLY  SM82  [CR]

PAGE 1                                          23:32:41  14 DEC 1982

SM82...........

WHERESUBS
A-CORR1
BRP1
IDATE
  .
  .
  .
```

The general forms of output-specification shown below will be discussed in the following sections.

```
attribute-name {"print limiters"}

BREAK-ON attribute-name {"text {'options'} text"}

TOTAL attribute-name {"total limiters"}
```

## 1.21 PRINT—LIMITERS

Print-limiters may be used to select only certain values from multivalued attributes for printing. The values can be limited to those satisfying specified criteria, and dependent values in associative data sets can be suppressed if the value they depend on is not printed.

Specific values from multivalued attributes can be selected for output by placing the relational operator(s), followed by the desired value (or values) in double-quotes (") or backslashes (\), immediately following the attribute name. If the attribute is an associative controlling attribute then the corresponding values from the dependent attributes will also be returned. Likewise, if the controlling value does not match any of the desired values in quotes, then the dependent values associated with those controlling values will not be printed.

For example, to limit the printing of an attribute called TRAN-DATE (which is a date field) to dates in the range 1/1/82 through 12/1/82 inclusive, the following print-limiting condition may be specified:

```
. . . TRAN-DATE > = \1 JAN 82\  AND <= "12/1/82" . . .
```

Note that the form of the print-limiter follows the general form used for selection-criteria; in fact, the only difference between a selection-criterion and a print-limiting output specification is the absence of the WITH modifier.

The following example lists all the items in the INV file which contain any value for the attribute TRAN-DATE:

```
>LIST INV TRAN-DATE TRAN-TYPE TRAN-QTY  [CR]

PAGE 1                                          18:18:36  20 AUG 1982

INV.........      TRAN-DATE   TRAN-TYPE   TRAN-QTY
                              *           *
1242-22           11 FEB      I           100
                              R            48
                              S            31
                  12 FEB      I           144
                              R            43
                              S            66
1242-11           11 FEB      I            19
                              R           122
                              S            33
                  12 FEB      I            97
                              R            39
                              S             7

2 ITEMS LISTED.
```

A search for a single value may be used in the print-limiter format. For example, the TRAN-DATE "11 FEB 82" portion of the ACCESS sentence following indicates to the processor that only the dates equal to 11 FEB 82 are to be retrieved.

```
<LIST INV TRANS-DATE "11 FEB 82" TRAN-TYPE TRAN-QTY  [CR]

PAGE 1                                          18:20:04   20 AUG 1982

INV....      TRAN-DATE   TRAN-TYPE   TRAN-QTY
                         *           *
1242-22      11 FEB      I           100
                         R            48
                         S            31
1242-11      11 FEB      I            19
                         R           122
                         S            33

2 ITEMS LISTED.
```

Since TRAN-DATE is a controlling attribute, only the values associated with 11 FEB 82 for TRAN-TYPE and TRAN-QTY (which are dependent attributes) are retrieved.

## 1.22 MODIFIERS AND OPTIONS

Modifiers or options may be used to further modify the meaning of ACCESS sentences, and to generate more elaborate listings and reports. Some modifiers appear before attribute names in the ACCESS sentence, and some appear at the end of a sentence and may be replaced by an option in the option string. The option string always appears last in an ACCESS sentence, and consists of single letters, optionally separated by commas, and enclosed in parentheses.

The modifiers which may be used in an ACCESS sentence, and their equivalent options, if any, are listed in alphabetical order below:

| Modifier | Equivalent Option | Meaning |
|---|---|---|
| BREAK-ON | | Defines control. (See Section 1.25, CONTROL-BREAKS.) |
| BY | | Designates the attribute name immediately following as a sort-key for the SORT operation. Sequencing is in ascending order comparing ASCII values. See Section 1.5, SORT. |
| BY-DSND | | Specifies sorting in descending instead of ascending order. |
| BY-EXP | | Sorts by exploding attribute values; ascending order. |
| BY-EXP-DSND | | Sorts by exploding attribute values; descending order. |
| COL-HDR-SUPP | (C) | Suppress the output of the time/date heading, the column headings, and the end-of-list message. |
| DBL-SPC | | Causes an extra blank line to be inserted between items to double-space a listing. |
| DET-SUPP | (D) | Suppresses detail output when used with TOTAL or BREAK-ON modifiers. See Section 1.25.2, GENERATING SUBTOTALS USING CONTROL-BREAKS. |
| DICT | | Specifies the DICT (dictionary) portion of a file is to be listed, as opposed to the DATA file. (See Section 1.1, FORMING ACCESS INPUT SENTENCES.) |

| Modifier | Equivalent Option | Meaning |
|---|---|---|
| EVERY or EACH | | Modifies a selection-criterion so that every value for a multivalued attribute must meet the specified condition for the criterion to be true for that item. This modifier must immediately follow the modifier WITH. See Section 1.18, FORMING SELECTION-CRITERIA.) |
| FOOTING | | Indicates that the following text string is to be processed and used for a footing on the bottom of each page of output. |
| GRAND-TOTAL | | Indicates the following text is to be printed on grand total lines. |
| HDR-SUPP or SUPP | (H) | Suppresses the output of the time/date heading and the end-of-list message. |
| HEADING | | Indicates that the following text string is to be processed and used for a heading on the top of each page of output. |
| ID-SUPP | (I) | Suppresses the display of item-ids for LIST and SORT operations. |
| LPTR | (P) | Routes output to line printer. |
| NOPAGE | (N) | When output is to the terminal, this modifier will prevent the halt of output at the end of each page. |
| ONLY | | Inhibits the use of default attribute definition items when no output specification is given. When used, must precede the file-name. |
| TAPE | | Causes the data to be obtained from a tape or cartridge disk file in a T-DUMP or S-DUMP format. |
| TOTAL | | Causes totals for the attributes which follow to be accumulated. |
| USING | | Causes the attributes to be obtained from a file other than the dictionary-level file. |
| WITH or IF | | Designates that the following attribute name is part of a selection criterion. |

| Modifier | Equivalent Option | Meaning |
|---|---|---|
| WITHIN | | Designates that items may contain additional item-ids relating to the primary item-id. |
| WITHOUT | | Is a synonym for WITH NOT or WITH NO. |

A, AN, ARE, ANY, FILE, FOR, IN, ITEMS, OF, OR, and THE are throwaway modifiers which do not affect the meaning of the ACCESS sentence. They may be used anywhere in the sentence and are included to provide a degree of naturalness to the language. Any other words not otherwise defined in an account's master dictionary may be included as throwaways by copying the definition of an existing throwaway to the new throwaway.

The following example demonstrates alternative ways of saying the same thing. In the first case, the terms 'THE' and 'FILE' are throwaways and may be included or omitted. The modifier DBL-SPC has no option form. Note that the default is to pause at the end of each page when the output is to a terminal. If you wish to terminate the listing at this point, type a CONTROL-X. If you prefer that the listing not pause at the end of each page, use the NOPAGE modifier or the N option.

```
>LIST THE ACCOUNT FILE DBL-SPC  [CR]

>LIST ACCOUNT DBL-SPC  [CR]
```

These sentences cause the ACCOUNT file to be listed. The listing will be double-spaced, and the output will halt at the end of each page.

```
>SORT INVENTORY WITH PRICE GT "500" ID-SUPP (P)  [CR]
```

This sentence causes items in the INVENTORY file with PRICE greater than 500 to be sorted and listed. The output will be to the line printer and the time/date heading will not be printed.

```
>LIST ACCOUNT ITEMS > "35000" NAME ADDRESS NOPAGE ID-SUPP  [CR]

>LIST ACCOUNT > "35000" NAME ADDRESS (IN)  [CR]
```

These sentences cause the values for NAME and ADDRESS (in items with item-ids greater than '35000') to be listed. Item-ids will not be listed. The listing will not halt at the end of each page.

```
>SORT-ITEM PROCS LPTR (F)  [CR]

>SORT-ITEM PROCS (P,F)  [CR]
```

These sentences cause all items in the PROC file to be output to the spooler, sorted by item-id, one item per page.

Note that 'SUPP' is a synonym for 'HDR-SUPP' and that the option H has the same effect.

The third case included the NOPAGE modifier, which causes a listing which is sent to a terminal to not pause at the end of each page. The option associated with this modifier is N. The option associated with the ID-SUPP modifier is I.

In the fourth case, note that the modifier LPTR may be replaced by the option P, and that the option F, which is applicable to the LIST-ITEM and SORT-ITEM verbs only, and which causes a page-eject to be executed before the beginning of each item, does not have a modifier item.

## 1.22.1 ACCESS AND THE OPTIONS PROCESSOR

Options are decoded during the initial instruction scan. The call to the options processor is initiated by the left parenthesis which must be used before the first option. The option processor will accept any uppercase alphabetic character. All other characters in the string will be ignored by the options processor. The following is a list of the options of use in ACCESS.

| Option | Meaning |
|---|---|
| B | Causes the line feed at the end of the compile phrase to be avoided. |
| C | Equivalent to COL-HDR-SUPP. Relevant to LIST-class verbs. |
| D | Equivalent to DET-SUPP. Relevant to verbs capable of BREAK-ON and TOTAL. |
| F | Causes page eject for each item. Relevant only to the LIST-ITEM and SORT-ITEM verbs. (No equivalent modifier.) |
| H | Equivalent to HDR-SUPP. Relevant to LIST-class verbs. |
| I | Equivalent to ID-SUPP. Suppresses the item-id in LIST-class verbs. Suppresses the item-id in the terminal output with T-LOAD, T-DUMP, and S-DUMP verbs. |
| N | Equivalent to NOPAGE. Relevant to terminal output with LIST-class verbs. |
| O | Overwrite items. Relevant to the T-LOAD verb. |
| P | Equivalent to LPTR. Relevant to LIST-class verbs. |

## 1.23 GENERATING HEADINGS AND FOOTINGS

The user can specify his own headings and footings with the ACCESS verbs LIST, SORT, LIST-ITEM and SORT-ITEM. This is done with the HEADING and FOOTING modifiers, which specify that the string that follows them is to be used as heading or footing text. Special characters inside the text string specify special operation on the heading. During generation of the listing, the special characters in the heading or footing text string will be replaced by the current value of the page number, the item-id of the item being listed, break-on data or other parameters. A provision for centering headings, even variable length ones, is provided. The specified heading or footing will be printed at the top or bottom of every page of output.

If an ACCESS input sentence contains a HEADING modifier, then the normal heading, which consists of a page number and the current time and date, and the usual "n ITEMS LISTED" message will not be printed.

A HEADING (FOOTING) specification may appear anywhere in the LIST type statement. To specify a heading, the user enters the word HEADING (FOOTING) followed by a string of characters enclosed in double quotes ("), or backslashes ( \ ). Special option characters may appear, enclosed in single quotes (´ ´). The HEADING (FOOTING) specification has the general form:

```
    HEADING or FOOTING "text {´options´} text {´options´}..."
```

Special option characters to be enclosed in single quotes are:

| Option | Meaning |
|---|---|
| ´B´ | BREAK. Inserts the value causing a control-break, if the ´B´ option has been specified along with the control-break field. See Section 1.25.3, OUTPUT OPTIONS FOR CONTROL-BREAKS). Has no effect otherwise. |
| ´C´ | CENTER. Causes the current line of the HEADING or FOOTING to be centered on the output page. |
| ´D´ | DATE. Inserts the current date at this point in the heading in the form: dd mmm yyyy, where dd is the day of the month, mmm is the abbreviation for the name of the month, and yyyy is the 4-digit year. |
| ´F´ | FILE-NAME. Inserts the name of the file being LISTed or SORTed. |
| ´Fn´ | FILE-NAME NO. #. (Where ´n´ is a decimal number.) Causes the file-name to be left-justified in a field of n blanks at that point in the HEADING or FOOTING. |
| ´L´ | LINE. Specifies the start of a new line in the HEADING or FOOTING. |

| Option | Meaning |
|---|---|
| ´P{n}´ | PAGE. Inserts the current page number, right justified in a field of 4 (or n) blanks. |
| ´PN{n}´ | As above, left justified. |
| ´T´ | TIME. Inserts the current time and date in the format: hh:mm:ss dd mmm yyyy, where hh is the hour in 24-hour (or "military") format, mm is the number of minutes and ss is the number of seconds past the hour, dd is the day of the month, mmm is the abbreviation for the name of the month, and yyyy is the 4-digit year. |
| ´´ | Two successive single quotes are used to print a single quote mark in heading text. |

For example:

```
    HEADING "INVENTORY LIST´C´"
```

which will center the heading "INVENTORY LIST" at the top of each page; or,

```
    FOOTING "´LC´ STATUS REPORT ´LC´ PAGE:´P´"
```

which will skip a line, center the message "STATUS REPORT", go to the next line and center the word "PAGE", with the current page number at the foot of each page. (The page number is printed right-justified in a field of 4 blanks.)

An example of HEADING usage is:

```
    >SORT ACCOUNT NAME HEADING "NAME LIST AT ´TL´ PAGE NO. ´PL´"  [CR]

    NAME LIST AT 10:29:39  20 AUG 1982
    PAGE NO.    1

    ACCOUNT...         NAME........

    11000              M H KEENER
    11015              L K HARMAN
    11020              J T O´BRIEN
    11025              P R BAGLEY
       .                   .
       .                   .
       .                   .
```

## 1.24 GENERATING TOTALS AND GRAND-TOTALS

LIST and SORT sentences may generate subtotals and totals for attribute values. The TOTAL modifier is used to generate sub- and grand-totals; the GRAND-TOTAL modifier is used to specify special formatting on the grand-total line. The general form of these modifiers is:

```
        . . TOTAL attribute-name {"total limiters"} . .

          . . GRAND-TOTAL "{text} {'options'} {text}" . .
```

The total modifier causes a total to be computed for the attribute whose name immediately follows the word "TOTAL". For example:

```
    LIST AFILE TOTAL A7
```

This sentence causes values for attribute A7 to be listed, followed by a total (sum) of these values. On the output, the default total identification is three asterisks (***) in the item-id column.

The total-limiter may be used to limit the total-ing to values that pass the limiting criterion. The form of the total-limiter is the same as that for the print-limiter (see Section 1.21, PRINT-LIMITERS), and the total-limiter will also cause a print-limiting function.

It is possible to total fields of length 0. Nothing will be printed at detail time. At break time, the value will appear at its appointed location, unless the output value is cleared to null with an F;"" in attribute 7 of the data definition item.

### 1.24.1 THE SEQUENCE OF EVALUATION

Totals are generated from the number that results after the execution of a correlative (attribute 8) of the data definition item, and after the test for the print- or total-limiter. If the result after the execution of attribute 8 is non-numeric, then the total will remain as it was before the current value was processed.

At output time, that is, on a control break or at the grand total, the data in the related control-break item and all of the totals are composed into an item. Each element of this item is obtained from the control-break or total record. The element is then used as input for the conversion in attribute 7 of the data definition item. Conventionally, this has the effect of masking the total according to the same MR-class mask as was used on the detail-time data in this field.

It is possible, however, to use F- or A-correlatives in attribute 7 to generate compositions of totals. In this case, the attribute-mark-count number specified in the F-correlative refers to the attribute-mark-count number in attribute 2 of the data defining items. In other words, if an F-correlative stipulates an AMC of 17, at detail-time, the processor will look in attribute 7 of a data item for the value requested. At total-time, the processor will look for the data defining item in the compiled process controlling string which has ´17´ as its AMC specifier. This is the number in attribute 2 of the data defining item in the dictionary file.

The processor will then retrieve the total related to that item at the current break level for processing in the F-correlative. It is, therefore, possible to generate data which is the result of arithmetic manipulations on totals.

It is in this context that the operand ´ND´ and ´NB´ (see Section 2.4.9.1, F-correlatives) are of use. The ND operand obtains the number of items processed since the last break at this level. (Note that BY-EXP SORTS and LISTS that are run using lists created with a BY-EXP modifier consider the collection of values obtained at each value mark to be an item.) The ND is useful for obtaining the number of values included in a total if there is a one-to-one correspondence between the values and the item count represented by ND. If there is not, a separate totaled data definition item whose definition includes F;"1" in attribute 8 should be included in the sentence and referenced in attribute 7 of the data definition item which is generating the composition at total times.

The NB operand returns the break level at which processing is currently occurring. NB is zero at detail-time, and is 255 at grand total time. The lowest level break yields 1; and for each succeeding break level the number is incremented by one. The processor is capable of at least 127 break levels, and may be capable of 250 levels. This means that a different class of number may be constructed at each break level.

The GRAND-TOTAL connective may be used in reports that have TOTALs and/or BREAK-ONs to specify special action when printing the grand-total line. The use of GRAND-TOTAL is similar to the BREAK-ON connective; the difference is that the GRAND-TOTAL connective may appear anywhere within the statement, and that it must be followed immediately by a value string enclosed in double-quotes.

The general form is:

```
    GRAND-TOTAL "...  text ... {´options´} ..{text}"
```

The optional "text" is any literal string that is to be printed as a substitute for the normal "***" that appears as the item-id of the GRAND-TOTAL line. The literal string will be printed left-justified, starting at column 1, regardless of the actual justification of the item-id, even if "ID-SUPP" has been used.

The 'options' string is enclosed within single-quotes, and is used to specify the 'U' (underline), 'L' (line-suppress), and 'P' (page-eject) options. The page-eject feature is particularly useful when the GRAND-TOTAL line of a report is actually meaningless; by specifying page-eject, the GRAND-TOTALs appear on a new page, and may be discarded. If the underline option is used, all total-ed fields in the report will be underlined with a row of equal-signs (=).

An illustration of the use of TOTAL and GRAND-TOTAL with an example from the ACCOUNT file is:

```
    >LIST ACCOUNT AFTER "35090" NAME ADDRESS TOTAL DEPOSIT  [cs] _ [CR]
    : GRAND-TOTAL "'U' GRAND-TOTAL IS :" [CR]

    PAGE 1                                          10:31:23  20 AUG 1982

    ACCOUNT... NAME..............       ADDRESS........  DEPOSIT.

    35100      R W FORSTROM             318 CARNATION        8.00
    35095      A W FEVERSTEIN           324 CARNATION       10.00
    35110      H E KAPLOWITZ            306 CARNATION       10.00
    35105      S J FRYCKI               312 CARNATION       10.00
                                                         ========
    GRAND TOTAL IS :                                        38.00

    4 ITEMS LISTED.
```

The TOTAL modifier may be used in conjunction with the BREAK-ON modifier to output subtotals. See Section 1.25.2, GENERATING SUBTOTALS USING CONTROL-BREAKS.

## 1.25 CONTROL-BREAKS

### 1.25.1 BREAK-ON MODIFIER

The BREAK-ON modifier may be used to group items in a listing according to the value of the BREAK-ON attribute-name(s). The BREAK-ON modifier is used in the following form:

```
    BREAK-ON attribute-name {"text...{'options'}..text"}
```

The attribute-name indicates the attribute on which a break will occur. The optional text string, if specified, will be printed instead of the normal break-on line. Options are provided to put additional information in the break-on text.

During the LIST or SORT operation, a control-break occurs whenever there is a change in the value of the specified attribute. Value comparison is made on a left-to-right, character-by-character basis. In generating the value for comparison, correlatives in the attribute definition are processed but conversions are not. See Section 2.4, CONVERSION AND CORRELATIVE CODES.

The hierarchy of the breaks is specified by the sequence of the BREAK-ONs in the input line, the first being the highest level.

When a control-break occurs, three asterisks (***) are displayed in the BREAK-ON attribute column (i.e., the attribute whose value has changed, thus causing the break), preceded and followed by blank lines. If the optional text string is specified, the processed text string will be substituted for the asterisks.

For multiple control-breaks, output proceeds from lowest level BREAK to highest level. The data associated with the lowest level control-break is printed on the current page (even if the end of the page has been reached). If multiple BREAKS occur, normal pagination proceeds on the second and subsequent data lines, unless an option prevents this.

The BREAK-ON modifier may be used in conjunction with the TOTAL modifier.

The data associated with the break-on attribute may be suppressed in the detail lines by using a MAX length of zero in the dictionary attribute definition (attribute 10 in the data definition item). If suppression of the data associated with the control break is desired at total time, it must be done with an F;"" in attribute 7 of the data definition associated with the control break.

The following example illustrates the use of the BREAK-ON modifier. Additional output formatting capabilities are described in Section 1.25.3, OUTPUT OPTIONS FOR CONTROL-BREAKS.

```
>SORT ACCOUNT > "35000" BY STREET NAME BREAK-ON STREET CURR-BALNC  [CR]

PAGE 1                                        09:34:01  02 APR 1982

ACCOUNT...        NAME.......       STREET.........  CURR-BALNC...

35090             D U WILDE         CARNATION        $      884.53
35095             A W FEVERSTEIN    CARNATION        $       19.25
35100             R W FORSTROM      CARNATION
35105             S J FRYCKI        CARNATION        $    5,569.53
35110             H E KAPLOWITZ     CARNATION        $   94,944.55

                                    ***

35005             J S ROWE          COVE             $      464.72-
35010             S R KURTZ         COVE             $      467.33
35015             W F GRUNBAUM      COVE             $       88.47
35025             J D GUETZINGER    COVE             $        3.45

                                    ***

35030             F M HUGO          DAHLIA           $      123.48
35035             M J LANZENDORPHER DAHLIA           $      445.89
35040             C E ESCOBAR       DAHLIA           $   38,822.12-
35055             P J WATT          DAHLIA           $      337.18
35055             J W ROMEY         DAHLIA           $   33,478.95

                                    ***

35060             J A SCHWARTA      DOCK             $   33,822.34
35065             L J RUFFINE       DOCK             $      558.43
35070             F R SANBORN       DOCK             $   22,144.67
35075             J L CUNNINGHAM    DOCK             $        7.70
35080             G A BUCKLES       DOCK             $  447,765.48
35085             J F SITAR         DOCK             $      200.00

                                    ***

***

20 ITEMS LISTED.
```

## 1.25.2 GENERATING SUBTOTALS USING CONTROL-BREAKS

The TOTAL modifier may be used with the BREAK-ON modifier for the purpose of generating subtotals in LIST and SORT statements when control-breaks occur. The GRAND-TOTAL modifier is used to print a heading or otherwise modify the grand TOTAL data line.

The TOTAL modifier is used to generate and print subtotal values (in addition to a total) when it appears in the same sentence as BREAK-ON. The form is the same as for generating totals, for example:

```
    TOTAL attribute-name {"total-limiter"}
```

Values for the specified attribute are accumulated and printed as subtotals whenever a control-break occurs. Multiple TOTAL modifiers may appear.

When a control-break occurs, a line of data is output, preceded and followed by blank lines. Three asterisks (***) are displayed in the BREAK-ON attribute column, and a subtotal is displayed in the appropriate column for each attribute specified in a TOTAL modifier. Subtotals are the values accumulated since the last control-break occurred.

At the end of the listing, a TOTAL line is printed for every BREAK used. All end of listing sums are printed on the current page.

In computing the value for accumulation, correlatives are processed but conversion specifications are not (see Section 2.4, CONVERSION AND CORRELATIVE CODES). Conversion is applied only when the value being accumulated is actually printed.

The GRAND-TOTAL modifier is used to control the format of the grand total line. Its general form is:

```
    GRAND-TOTAL "text...{'options'}...text"
```

The text, if specified, replaces the "***" field normally printed at the extreme left of the GRAND-TOTAL line. Output formatting capabilities are described in Section 1.25.3, OUTPUT OPTIONS FOR CONTROL-BREAKS.

Sample use of control-breaks:

```
>SORT ACCOUNT WITH BILL-RATE "2" "40" NAME BREAK-ON BILL-RATE  [cs]_ [CR]
:TOTAL  CURR-BALNC BY BILL-RATE  [CR]

PAGE 1                                          09:28:03  23 AUG 1982

ACCOUNT...        NAME..........    BILL-RATE       CURR-BALNC..

35060             J A SCHWARTA          2           $ 33,822.34
35085             J F SITAR             2           $    200.00

                                      ***           $ 34,022.34

11100             E F CHALMERS         40           $     17.50
35075             J L CUNNINGHAM       40           $      7.50

                                      ***           $     25.20

***                                                 $ 34,047.54

4 ITEMS LISTED.
```

## 1.25.3 OUTPUT OPTIONS FOR CONTROL-BREAKS

Headings and output control options may be specified for control-breaks. The DET-SUPP modifier may be used to suppress detail in listings.

A user-generated heading can be specified to be printed in place of the default name CONTROL-BREAK HEADING (***) by following the BREAK-ON attribute-name with the desired heading, enclosed in double quote marks (" "). Within the heading, output control options may be specified, enclosed in single quote marks (´ ´). This gives the BREAK-ON specification the following form:

```
BREAK-ON attribute-name {"text...{´options´}...text"}
```

The text, if specified, replaces the default asterisk field ("***") in the attribute-name column when the control-break printout line occurs. Options are used to modify some of the actions taken at control-break time, and are specified as one or more characters. BREAK-ON options are:

| Option | Meaning |
|---|---|
| ´B´ | BREAK. Specifies this control break attribute name as the one whose value is to be inserted in the ACCESS page heading in place of the ´B´ option in the HEADING specification (see GENERATING HEADINGS). It may not be meaningful to specify this option within more than one BREAK-ON specification. |
| ´D´ | DATA. Suppresses the break data line entirely if there was only one detail line since the last time this control-break occurred. |
| ´L´ | LINE. Suppresses the blank line preceding the break data line. This option is ignored when the ´U´ option (see below) is used. |
| ´N´ | NUMBER. Resets the page number to one on this break. |
| ´P´ | PAGE. Causes a page eject after the data associated with this break has been output. |
| ´R´ | ROLLOVER. Inhibits page rollover, thus forcing all the data associated with this break to be current on the same page. |
| ´U´ | UNDERLINE. Causes the underlining of all specified TOTAL fields. |
| ´V´ | VALUE. Causes the value of the control-break to be inserted at this point in the BREAK-ON heading. |

If the modifier DET-SUPP is used in the sequence with TOTAL and/or BREAK-ON, then all detail will be suppressed and only the subtotal and total lines will be displayed. This is shown in the second ACCESS sentence of the following example:

```
>SORT ACCOUNT WITH BILL-RATE "2" "40" BY BILL-RATE NAME [cs]_[CR]
:BREAK-ON BILL-RATE "SUB-TOTAL FOR 'V'" TOTAL CURR-BALNC  [cs]_[CR]
: GRAND-TOTAL "GRAND TOTAL:  'U'"  [CR]

PAGE 1                                          11:22:33  22 FEB 1982

ACCOUNT...        NAME..........BILL-RATE        CURR-BALNC..

35060             J A SCHWARTA       2           $  33,822.34
35085             J F SITAR          2           $     200.00

                    SUB-TOTAL FOR 2              $  34,022.34

11100             E F CHALMERS      40           $      17.50
35075             J L CUNNINGHAM    40           $       7.70

                    SUB-TOTAL FOR 40             $      25.20
                                                 ============
GRAND-TOTAL:                                     $  34,047.54

4 ITEMS LISTED.

>SORT ACCOUNT WITH BILL-RATE "2" "40" BY BILL-RATE NAME [cs]_[CR]
:BREAK-ON BILL-RATE "SUB-TOTAL FOR 'V'" TOTAL CURR-BALNC DET-SUPP [CR]

PAGE 1                                          09:39:20  20 AUG 1982

ACCOUNT...        BILL-RATE         CURR-BALNC..

    SUB-TOTAL FOR         2         $  34,022.34

    SUB-TOTAL FOR        40         $      25.20

***                                 $  34,047.54

4 ITEMS LISTED.
```

# data entry 2

## 2.1 ACCESS DICTIONARIES AND ATTRIBUTE-DEFINITION ITEMS

The information in sections 2.1 through 2.4 deals with getting information into the ACCESS data base. This documentation is more technical and is provided for the person who wishes to write a program for data entry into ACCESS. Another way to get data into ACCESS is to create the dictionary and associated files structures described using the EDITOR. Although this is not an efficient method of data entry, it gives the newcomer to PICK a simple way to experiment with ACCESS data entry and become familiar with the operation of PICK at the same time.

ACCESS uses dictionary items to define the data structure in the file. Each data file has an associated dictionary, which may contain a set of items used to define the data. The item-ids of these dictionary items are the "attribute-names" used in ACCESS statements in selection-criteria, sort-specifications, and other specifications. These names may be of any form and length, but must not be the same as any of the modifiers and connectives that are defined in the MD and are, therefore, reserved words. For example, in the statement:

```
>LIST ACCOUNT WITH CURR-BALNC NAME CURR-BALNC LPTR [CR]
```

| | |
|---|---|
| ACCOUNT | is the file-name |
| WITH | is a modifier and is in the MD |
| CURR-BALNC and NAME | are items in the ACCOUNT dictionary and, therefore, attribute-names |
| LPTR | is a modifier and is in the MD |

Each dictionary item serves to:

- Define the location of the data field within the data item.
- Define a "tag" or heading field for output.
- Define interrelationships between attributes.
- Define output formats, table look-ups, etc.

The dictionary items that define the data format for ACCESS processing are called "attribute-definition" items. The item-id is the attribute-name which may or may not be the same as the tag in line 3. Line 1 of an attribute-definition item contains the code "A" (therefore, these dictionary items are also called "A" items). Other lines contain data as described below. Note that "line" refers to a line in the EDITOR, and that PICK considers each new line to be a new attribute. Therefore, when creating the PICK data files with the EDITOR, line 1 will contain the data for attribute-name1, line 2 the data for attribute-name2, and so on.

| Line | Name | Description |
|---|---|---|
| 1 | D/CODE | A, S or X. Defines attribute-definition item. "A" indicates "attribute," "S" indicates synomym attribute," and "X" means "skip this attribute now". |
| 2 | A/AMC | Numeric value giving the location of data defined by this item (attribute mark count). May contain 0 if referencing the item-id, or a dummy value if.the data referenced is computed or generated, but not actually stored. It is used to identify controlling and dependent attibutes. Also, an amc of 9999 is used to access the SIZE or count field of the item; an amc of 9998 to access the current item counter (item sequence number). |
| 3 | S/NAME | Textual data used as tag to appear in headings in LIST or SORT statements. If null, the item-id is used as the tag. May contain blanks for formatting purposes. The reserved character " " is used to specify a null tag. Multiple line tags for COLUMNAR listings only may be specified by storing multiple values (separated by a value-mark, control-]) in this field. |
| 4 | STRUCT | This defines the "controlling-dependent" relationship. |
| 5,6 | | Not Used. |
| 7 | V/CONV | This is the Conv attribute; it contains the conversion specification(s) which is (are) used to convert from the processing format to the external (displayed) format. Multiple specifications are separated by value-marks (control-]). |
| 8 | V/CORR | This is the Corr attribute; it contains the correlative specification(s) which is (are) used to convert from the internal format to the processing format. Multiple specifications are separated by value-marks (control-]). |
| 9 | V/TYP | This defines the justification (left or right). An entry may be an "R", "L", "T", or "U". This code is used both in formatting the output and in determining the sort sequence when sorting data. An "R" is used to specify a right-justified numeric sort (even for alphanumeric fields). An "L" will always sort left to right and will left-justify, folding at the end of the field. A "T" will left-justify and, if the value exceeds the specified maximum length, will fold at blanks. A "U" code causes left-justification without foldings. V/TYP defaults to ´L´ if it is null. |
| 10 | V/MAX | This defines the maximum length of values for the attribute. The entry is a decimal number. A value of 0 may be used to suppress the listing of a control-break field on detail lines. V/MAX defaults to 10 if it is null. |

Typically, there are several dictionary items that can refer to a particular data field in the file, since different formatting, sorting, or selection requirements may require them. Multiple items are commonly called "synonym" items and there is no limit to the number of such synonyms. For example, you may also want to sort a field using a right-justified (numeric) sorting sequence, but may want to output the data left-justified, which requires two different dictionary items. An example of such statements:

1. `>SORT INVENTORY BY PART# PART# [CR]`

2. `>SORT INVENTORY BY ONE ONE [CR]`

Sample PART#'S

```
P2000-99C
P2000-12A
P2000-105B
```

```
Sorted output (statement 1)                    (statement 2)
                    P2000-12A                  P2000-105B
                    P2000-99C                  P2000-12A
                   P2000-105B                  P2000-99C
```

Dictionary items in INVENTORY FILE:

```
Item-Id    PART #                ONE
Line-No    001 A                 001 A
           002 1                 002 1
           003 PART#             003 ONE
           004                   004
           005                   005
           006                   006
           007                   007
           008                   008
           009 R                 009 L
           010 10                010 10
```

## 2.2 ACCESS AND THE FILE STRUCTURE

ACCESS files are made up of elements found in files and values related to the actual data to be retrieved. The verb definitions, file definitions, modifiers, and relational operators must be in the MD. Usually, the data will be found in the data section of the file and attribute-definition items in the dictionary of the data file. However, other alternatives are available.

### 2.2.1 THE USING CONNECTIVE

The dictionary look-up process may be directed to a file other than the file dictionary by use of the USING connective. The USING connective allows you to specify what file is to be used as the dictionary in the ACCESS sentence.

This explicit dictionary file may be either a dictionary-level file, specified by the modifier DICT, or a data-level file.

The USING statement may be anywhere in the sentence. It will be used for all data definition item look-ups executed by the ACCESS compiler.

The USING modifier may be used when retrieving a dictionary-level file.

The form of the USING phrase is:

```
USING DICT FILENAME

USING FILENAME
```

The first form references the dictionary of the file FILENAME; the second references the data-level file FILENAME. Note that the data-level file may be of the form FILENAME,SUBFILENAME. In these cases, all data definition items will be taken from the file referenced by the USING connective. Any items not found, will be looked for in the MD.

Only one USING connective is allowed in the ACCESS sentence. The USING connective must be immediately followed by either DICT FILENAME or FILENAME. The source of the data processed is still specified by the conventional file-name which must also appear in the sentence. Q-pointers may be used for either or both references.

Default data definition items will come from the file referenced by the USING connective only. Some examples are:

| | |
|---|---|
| LIST WORKFILE USING DICT TESTDICT | The data source will be WORKFILE. The data definition items will be retrieved from the dictionary of TESTDICT. The default data definition items will be used. |
| LIST DICT WORKFILE USING DICT WORKFILE | The data and data definition items have the same source. |
| LIST WORKFILE USING WORKFILE, WFDICT1 | The data comes from WORKFILE. The data definition items come from a data-level subfile of WORKFILE named WFDICT1. There may be an indefinite number of these. |

### 2.2.2 THE MASTER DICTIONARY DEFAULT

If the data definition item is not found in the dictionary specified for the file, in the dictionary-level file if no USING connective is included in the sentence, or in the file specified by the USING connective, then the ACCESS compiler will search the MD for the data definition item and will include it if found.

### 2.2.3 THE SEQUENCE OF RETRIEVAL OF ITEMS FROM FILES

The ACCESS compiler takes two passes to retrieve all the definitions from files which it needs to execute a sentence. The first pass uses the MD to find the file names and all modifiers and relational operators in the sentence. Any data definition items found in the MD will be ignored on this pass. When the input string is exhausted, the compiler proceeds to look up all undefined terms in the dictionary-level file either implicitly defined by the sentence or explicitly defined by the USING connective. Items which are found are included in the string in the proper location. If an item is not found in the specified dictionary, then the compiler will look it up in the MD. If it still does not find the item in the MD, it will concatenate a blank and the next data definition item-id in the string to the missing item-id. The compiler will attempt to look up this new key in the dictionary-level file and the MD. This process will terminate either when a data definition is retrieved or the list of data definition items is exhausted. If an item cannot be found, ERRMSG 24 THE WORD 'A' CANNOT BE IDENTIFIED, will be issued.

The compiler does not look up elements in the string which are enclosed in double quotes, single quotes, or backslashes. These are taken to be literals rather than variables. They have the effect of terminating a string of data definition item-ids.

### 2.2.4 ITEM-ID DEFINITION WITH Q-POINTERS TO THE DATA FILE

The file definition item in an ACCESS sentence allows the use of attributes 7, 9, and 10 as meaningful elements in the data definition. The label comes from the D- or Q-pointer name because attribute 3 of the D- or Q-pointer is, in general, otherwise occupied. Attribute 2 is obviously forced to 0. Note that the selection, sort, and output processors all ignore attribute 8. The selection and sort processors take the item-id as it is; the output processor allows the use of an attribute 7 conversion.

The justification is of importance to both the sort and output processors; the field length is of importance to the output processor (especially if the item-ids are significantly longer then the file name or its nominal field length), and especially to the columnar processor. Note that item-ids do not fold, unlike other data processed by the columnar processor. All of the characteristics of item-id handling may be got around by using a data definition item which references data attribute 0 with ID-SUPP.

In the case that a Q-pointer is used to reference a file, the contents of attributes 7, 9 and 10 in the Q-pointer definition take precedence over those attributes in the D-pointer if they exist in the Q-pointer. Those that do not exist in the Q-pointer will be retrieved from the D-pointer. In the case that they do not exist in either, attribute 7 will be defined as null, attribute 9 will become L, and attribute 10 will become 9. These defaults will be taken for all data definition items processed by the ACCESS compiler.

This allows the creation of multiple Q-pointers which treat the item-id field in different ways, as may be convenient. It remains that Q-pointers require at most that the first three attributes be defined.

### 2.2.5 ITEM-ID STRUCTURE

In TCL, the universal delimiter is a blank. Verbs, file names, connectives, and data definition names are normally delimited by blanks. Non-ACCESS verbs which reference files and items will take a string of characters which is delimited by blanks to be the file name or one of the item names, depending on its location in the command sequence.

If one constructs an ACCESS sentence which references a data definition item which the processor cannot find in either the specified file dictionary or the MD, it will then generate another item-id by taking the item-id for which a record did not exist and concatenating the next string delimited by blanks in the sentence to it, with a blank between the two character strings. This will now be used as an item-id. This is why the error message which is trying to tell you that the data definition item is not on file may include elements of the ACCESS sentence.

For example, if, in the example below, the data definition item DOG is not on file.

```
    LIST MD CAT DOG RAT
```

The error message:

```
    [24]  THE WORD "DOG RAT" CANNOT BE IDENTIFIED
```

will be returned.

The sequence of concatenated strings will terminate at the end of the sentence or at the first connective, value, or item-id which succeeds the unidentified word.

This means that a blank is, in general, a character which is allowable for item-ids in the system. If you wish to EDIT an item-id which includes one or more blanks, enclose the string in one of the value delimiters.

You may also use the delimiters within a string enclosed in delimiters. Simply use a delimiter which is not part of the item-id as the value-surrounding delimiter. For example:

IF DOG RAT is an attribute-definition item in MD, then

```
    LIST MD DOG RAT          Will return the one attribute-definition
                             item whose name is DOG RAT.
```

In order to modify the item DOG RAT, use the form:

```
    EDIT MD "DOG RAT"        Which will obtain the item.
```

If you have an item named O'HARA, use the form:

```
    LIST MD "O'HARA"         This will return the item O'HARA.
```

Similarly, the form:

```
  SELECT CUSTOMERFILE WITH LASTNAME "O'HARA"   Will find all the O'HARAs
                                               in the file CUSTOMERFILE.
```

There are characters which are not acceptable for inclusion in item-ids. None of the system delimiters is allowable. An attempt to retrieve an item-id which contains a system delimiter will have the effect of terminating the key at the delimiter. In other words, if you have an item-id with a system delimiter in it, it can be accessed only by processors which use a sequential search rather than retrieval by item-id. This means that you can list it or select it, but you cannot Edit it, Copy it, Delete it, or otherwise update it, or retrieve it using a list. It can be removed by using a file-to-file copy, since it will not copy to the new file. The fragment of the errant item-id up to the delimiter will be reported as not being on file at the end of the copy process if the copy is driven from a selected list.

It is also not advisable to include the control characters X'00' through X'1A' in the item-id.

## 2.3 CONTROLLING AND DEPENDENT ATTRIBUTES

There is the facility in ACCESS to define a set of attributes that are associated together for listing or other purposes. Such an associative set of attributes have one "Controlling" attribute; the other attributes in the set are called "Dependent."

An associative set of attributes is used where there is a definite relationship between the attributes, and the attributes are typically multivalued. The "Controlling" attribute has multiple values; associated with each one of these multivalues is a corresponding set of values in each of the dependent attributes. The dependent attributes may in turn have sub-multi-values; however, each set of submultivalues is considered one value for associative purposes.

The controlling-dependent set must be maintained in a particular format by whatever program is updating the file; see the sections on file structure in the Introduction to PICK manual for a general discussion on the physical item format and the use of value delimiters.

The controlling attribute may be used in several ways to improve formatting or to limit or control the data output.

For example, if the controlling attribute of an associative attribute set has a print-limiter on it, the dependent attributes will automatically be limited also. That is, if only the first and the seventh multivalues of the controlling attribute pass the print-limiting test, only the first and the seventh multivalues of all associated dependent attributes will be output. In columnar formats, there may be a blank line output when print-limiters are so used.

The controlling-dependent attributes are specified by the "C" and "D" structure codes in attribute 4 of the dictionary items. This is described in the next section.

There may be more than one associative attribute set in a file.

Dependent attributes may not be specified in an output specification without the controlling attribute also being specified. However, for special purposes, a synonym attribute definition without the "D" code may be used for listing the dependent attribute data by itself.

As an example, consider the system ACC file, which contains the accounting history data. Attribute 4 of this file stores the dates that the user has logged on; redundant values are not stored; that is, if a user logs on more than once in the same day, only one date is stored. Thus, values in this attribute are unique. This attribute is considered the Controlling attribute of the associative attribute set.

Attributes 5, 6, 7 and 8 store the actual time that the user logged on, the connect-time associated with each logon session, the number of CPU units used, and the number of line printer pages printed, respectively. Since there may be more than one of these sets associated with a particular date logged on, these are stored as submultivalues.

For example, a listing of an item "SMITH#0" in the file may look like:

```
    >LIST ACC "SMITH#0" (H)  [CR]

    ACC..............        DATE....    TIME....     CONN... UNITS LPTR
                                         *            *         *    *
    SMITH#0                  03/01/78    10:33AM      00:10     222  11         1
                                         04:15PM      00:05      23             1
                             03/03/78    11:12AM      00:33    1123  78         2
                                         12:13PM      00:04      34             2
                                         01:01PM      01:03    3090   6         2
```

There are two dates in the Controlling attribute, 03/01/78 and 03/03/78; associated with these two dates are two sets of values in the attributes TIME, CONN, UNITS and LPTR. (These are marked as "1" and "2" on the far right.) The data under the DATE column is stored as two multivalues; data under the other columns is also stored as two multivalues, but each is further comprised of submultivalues; two in the first set and three in the second.

The internal format of the item SMITH#0 is shown below; the representation here is not actual, since dates and times are actually stored in an internal format, but they are shown as if stored in the format displayed below, for clarity. The symbols [am], [vm] and [svm] stand for attribute mark, value mark and subvalue mark, respectively.

```
    SMITH#0[am][am][am][am]03/01/78[vm]03/03/78[am]
     .. Attribute 4.......

    10:33AM[svm]04:15PM[vm]11:12AM[svm]12:13PM[svm]01:01PM[am]
     .. Attribute 5 ......................................

    00:10[svm]00:05[vm]00:33[svm]00:04[svm]01:03[am]
     .. Attribute 6 ........................

   222[svm]23[vm]1123[svm]34[svm]3090[am]
    .. Attribute 7 ......................

    11[vm]78[svm][svm]6[am]
     .. Attribute 8 ..........
```

## 2.3.1 MORE ON CONTROLLING (C) AND DEPENDENT (D) ATTRIBUTES

'C' and 'D' structure code operators define controlling and dependent attributes.

Attribute 4 of attribute definition items is reserved for 'C' and 'D' structure definition codes. A 'C' code indicates a controlling attribute; a 'D' code indicates a dependent attribute.

In order to be a controlling attribute, the attribute definition item must have a 'C' code in line 4. This code must be followed by the attribute numbers of all dependent attributes for the controlling attribute.

The general form of the 'C' (controlling) structure definition code is:

```
    C;amc;amc;...amc
```

where 'C' is the capital letter C, and each 'amc' is a dependent attribute number. For example, if attribute 30 was to be a controlling attribute for (dependent) attributes 31, 32, and 33, then the attribute definition item for attribute 30 would have the following 'C' code in attribute 4:

```
    C;31;32;33
```

This indicates a controlling attribute which controls dependent attributes 31, 32, and 33. Each of these dependent attributes must have a dependent structure-definition code (a 'D' code) in attribute 4.

The general form of the 'D' (dependent) structure-definition code is:

```
    D;amc
```

where 'D' is the capital letter 'D', and 'amc' is the single attribute number of the attribute which controls the dependent attribute. In the earlier example, the attribute-definition items for attributes 31, 32, and 33 would have the following 'D' code in attribute 4:

```
    D;30
```

In the following example, attribute 1, the check number, controls dependent attributes 2, the check amount, and 3, the check date. In the first ACCESS sentence, all values for each attribute name are listed. In the second sentence, the user only wishes to see data on check number 502, so he uses a print limiter on the attribute name CK-NO. Since the attributes named CK-AMOUNT and CK-DATE are dependent upon the attribute named CK-NO, only the check amount and date for check number 502 are printed.

The dictionary of the CHECK-REGISTER file looks like:

| item-id | CK-NUMBER | CK-AMOUNT | CK-DATE |
|---|---|---|---|
| 001 | A | A | A |
| 002 | 1 | 2 | 3 |
| 003 | Check Number | Amount | Date |
| 004 | C;2;3 | D;1 | D;1 |
| 005 | | | |
| 006 | | | |
| 007 | | MR2$ | D |
| 008 | | | |
| 009 | R | R | L |
| 010 | 3 | 12 | 15 |

The item with item-id "JAN" in the CHECK-REGISTER file looks like:

| item-id | JAN | |
|---|---|---|
| 001 | 501[vm]502[vm]503 | (Multivalued check numbers) |
| 002 | 12300[vm]400[vm]4488 | (Multivalued check amounts) |
| 003 | 3289[vm]3291[vm]3291 | (Multivalued check dates) |

```
>LIST CHECK-REGISTER "JAN" CK-NUMBER CK-AMOUNT CK-DATE ID-SUPP  [CR]

CHECK-REGISTER CHECK NUMBER AMOUNT...... DATE............

JAN            501                 $123.00 01 JAN 1977
               502                   $4.00 03 JAN 1977
               503                  $44.88 03 JAN 1977

>LIST CHECK-REGISTER "JAN" CK-NUMBER "502" CK-AMOUNT CK-DATE ID-SUPP  [CR]

JAN            502                 $4.00 03 JAN 1977
```

## 2.4 CONVERSION AND CORRELATIVE CODES

Processing codes may be specified as either CORRELATIVE codes or CONVERSION codes, depending upon when the user wants the codes to be applied to the data.

An ACCESS attribute-name dictionary item may specify a processing code either in line 7, where it is called a CONVERSION code, or in line 8, where it is called a CORRELATIVE code.

During execution of an ACCESS sentence, the data in the items being listed is represented in three different formats. The first is the "stored" format, which is the format of the attributes exactly as they appear in the items in the file. Whenever a piece of data is retrieved from a file, it is picked up in stored format.

The CORRELATIVE code, if any, may then be applied to the data, converting it to "intermediate" format. The intermediate format is used whenever:

1. The attribute name is part of a sort key.
2. The data is compared to a selection criterion.
3. The attribute name has a print limiter.
4. The attribute name has a TOTAL or GRAND-TOTAL connective.
5. The data produces a control break.
6. The data is printed, except on break lines.

CONVERSION codes are applied as output conversions to the intermediate format data whenever the data is printed, including the break lines. This transforms the data from "intermediate" format to "external" format. The data is printed in external format.

If a CONVERSION is specified for an attribute name which is followed by a selection-criterion, the conversion is applied as an input conversion to the values in the ACCESS sentence to form the selection criterion value.

Conversions and correlatives may be multivalued, in which case they are separated by a value-mark (vm = ] = hex 'FD').

A summary of conversion and correlative codes is provided in Table 2-1.

Table 2-1. Conversion and Correlative Code Summary

| Name | Description |
|---|---|
| A | ARITHMETIC. Used to compute mathematical expressions. Converted to an "F" code at run-time. |
| C | CONCATENATE. Used to concatenate attribute values. |
| D | DATE. Used to convert dates to external format. |
| F | FUNCTION. Used to compute a mathematical function on attribute values |
| G | GROUP. Used to extract one or more fields separated by a given delimiter. |
| L | LENGTH. Used to place constraints on what kind of data will be returned, based on the length. |
| MC | MASK CHARACTER. Used to convert strings to upper and lowercase, or to extract alphabetic or numeric characters from strings. |
| ML | MASK DECIMAL. (Left justified) used to format and scale numbers and dollar amounts (same as BASIC format string). |
| MR | MASK DECIMAL. (Right justified) Same as ML. |
| MT | MASK TIME. Used to convert time of day from internal to external format. |
| MX | MASK HEXADECIMAL. Used to convert ASCII character strings to their hexadecimal (base 16) representations. |
| P | PATTERN MATCH. Used to return only those data values which match the specified pattern. |
| R | RANGE. Used to return only those data values which fall within the specified ranges. |
| S | SUBSTITUTION. Used to substitute the data value for non null or zero values. |
| T | TEXT EXTRACTION. Used to extract a fixed field from an attribute value. |
| Tfile | FILE TRANSLATION. Used to convert attribute values by translating them through another file. |
| U | USER DEFINED. Used to evoke assembly language routines to perform special user-written conversions or correlatives. |

## 2.4.1 GROUP EXTRACTION CODE (G)

The ´G´ code is used to extract one or more fields from an attribute value.

If an attribute value consists of multiple fields separated by a delimiter, the ´G´ code will extract one or more contiguous fields. The general form of the ´G´ (Group Extraction) code is:

```
G{m}*n
```

where:

| | |
|---|---|
| G | is the capital letter G (the Group Extraction code) |
| m | specifies the number of fields to skip. If m is not specified, zero is assumed and no fields are skipped. |
| * | represents any single non-numeric character, except a minus sign (-), which is the field separator. System delimiters should not be used. |
| n | is a decimal number which is the number of contiguous fields to be extracted. |

Sample usage of ´G´ code:

| G Code | Attribute Value | Output Value |
|---|---|---|
| G/1 | 04/02/1956 | 04 |
| G1/1 | 04/02/1956 | 02 |
| G2/1 | 04/02/1956 | 1956 |
| G/2 | 04/02/1956 | 04/02 |
| G1/2 | 04/02/1956 | 02/1956 |
| G0*1 | 123*888*444 | 123 |
| G1*2 | 123*888*444 | 888*444 |
| G2*1 | 123*888*444 | 444 |
| G3*1 | 123*888*444 | (null) |
| G1*1 | *WRITTEN 21 DEC 1977 | WRITTEN 21 DEC 1977 |

## 2.4.2 OUTPUT CONSTRAINT CODES (L AND R)

The ´L´ and ´R´ codes place restrictions on output based on the R(ange) and the L(ength) of a value.

The Length code places length constraints on what kind of data will be returned, or returns the length of the data. The user may select a fixed length Ln (number of characters) necessary to meet output criteria, or a length range Ln,m.

Examples of the L code are:

| | |
|---|---|
| L0 | Returns the length of the data string submitted to the length processor. |
| Ln | Returns the data value if it is equal to n characters long. If the data does not meet the criteria then null is returned. |
| Ln,m | Returns the data value if it is greater than or equal to n characters long and less than or equal to m, where n<m. |

The Range code returns data values which fall within the specified ranges, where multiple ranges are allowed.

The format of the Range code is:

| | |
|---|---|
| Rn,m{;n,m...} | Returns the data value if it is greater than or equal to n and less than or equal to m. Multiples of ranges may be specified as shown, in ascending order. Note that when negative range sets are used, the more negative number must be stated first. |

Note that any delimiter (except system delimiters) may be used to separate the numbers in a range. However, for the sake of clarity, a minus sign should not be used, as the minus sign may refer to the number(s) in the range.

In all cases, if the range(s) specifications are not met, null is returned.

## 2.4.3 OUTPUT CONSTRAINT CODES (P AND S)

The 'P' and 'S' codes place restrictions on output based on P(attern) matching and S(ubstitution) of the value of an attribute.

When an attribute is requested, the Pattern Match code returns only those data values within the attribute which match the specified pattern. If the data does not match the specified pattern, then null is returned. The Pattern Match consists of any combination of the following:

| | |
|---|---|
| P(nN) | An integer number followed by the letter 'N', which tests for that number of numeric characters. |
| P(nA) | An integer number followed by the letter 'A', which tests for that number of alpha characters. |
| P(nX) | An integer number followed by the letter 'X', which tests for that number of alphanumeric characters. |
| P('literal') | A literal string, which tests for that literal string. |

For example, if the user wished to have only social security numbers returned, the following Pattern Matching code could be used:

```
P(9N);(3N-2N-4N)
```

This specifies that only those data values comprised of 9 numeric or a string match of 3 numeric, a hyphen, 2 numeric, a hyphen, 4 numeric are returned.

The Substitution code substitutes the data value of the referenced attribute with a specified attribute if the data value is not null or zero. If the data value is null or zero, it will be substituted with a literal string specified by the user. For example:

```
S;4;'XXX'
```

This specifies that if the data value is not equal to zero or null then it will be replaced by the contents of attribute 4. If it is equal to zero or null, it will be replaced by the string 'XXX'. Note that if the Substitution code is used in conjunction with the Function Processor, it may serve to test for null or zero, and take different actions according to what kind of data it encounters. For example:

```
F1(S;*;'NORMAL VALUE')
```

This specifies that if attribute 1 is null or zero, the string "NORMAL VALUE" will be used, otherwise the contents of attribute 1 will be used.

Note that the S code should only be used in attribute 8.

## 2.4.4 CONCATENATIONS CODE (C)

The 'C' code provides the facility to concatenate attributes and/or literal values prior to output. The general form of the 'C' code is:

```
    C x op x {op x op...}    C op x op {x op x...}
```

where:

| | |
|---|---|
| C | is the code name. |
| x | is the character to be inserted between the concatenated attributes and/or literals. A semicolon (;) is a reserved character that means no separation character is to be used. Any non-numeric character (except a system delimiter or an asterisk) is valid, including a blank. |
| op | is an attribute mark count (amc) or any string enclosed in single quotes ('), double quotes (") or backslashes (\), or is an asterisk (*) which specifies the last generated value (from a previous Conversion or Correlative operation). |

Examples of the Concatenate code are:

```
    C10;"  ";11
```

which will concatenate the values from attributes 10 and 11, separated by two spaces.

```
    C2-*
```

which will concatenate the value from attribute 2, to a previously generated value from a preceding conversion or correlative code.

A safe practice for getting correct results is to create a dummy attribute dictionary item with AMC=0. Place your C code in this.

Sample usage of 'C' (Concatenate) code:

| ATTRIBUTE VALUES | C CODE | RESULTANT OUTPUT |
|---|---|---|
| 014 SMITH<br>015 JOHN H. | C;"NAME":14,15 | NAME:SMITH,JOHN H. |
| 001 DIME<br>002 DOZEN | C;1/2 | DIME/DOZEN |

## 2.4.5 TEXT EXTRACTION CODE (T)

The 'T' code is used to extract a fixed number of contiguous characters from an attribute value. This is useful for fixed field data, or for truncating data when you wish to prevent folding. Note that the attribute value that contains the characters to be extracted must not exceed 500 bytes. The general form of the 'T' (Text Extraction) code is:

```
T{m,}n
```

where:

| | |
|---|---|
| T | is the code name |
| m | is the optional starting column number |
| , | is the separator necessary between 'm' and 'n' if 'm' is specified |
| n | is the number of characters to extract |

If the form 'Tn' is specified, 'n' characters will be extracted, either from the left or the right, depending upon the attribute definition item's V/TYPE (attribute 009). If the V/TYPE is 'L', the 'Tn' form code will extract the first 'n' characters of the attribute value. If the V/TYPE is 'R', the 'Tn' code will extract the 'n' rightmost characters of the attribute value.

If the form 'Tm,n' is specified, then 'n' characters, starting at column 'm', will be extracted. The 'Tm,n' form always counts columns and extracts characters from left to right. This means that the extraction will be from left to right, regardless of the attribute definition item's V/TYPE. Examples of T code:

| T CODE | ATTRIBUTE VALUE | JUSTIFICATION | VALUE OUTPUT |
|---|---|---|---|
| T3 | ABCDEF | L | ABC |
| T3 | ABCDEF | R | DEF |
| T3,5 | HELLO OUT THERE | L | LLO O |
| T3,5 | HELLO OUT THERE | R | LLO O |
| T1,11 | THIS IS A LONG STRING | L | THIS IS A L |
| T2,9 | *12 DEC 77 | L | 12 DEC 77 |
| T4,7 | 123SMITH  CR | L | SMITHbb * |
| T4,7 | 848JOHNSONDB | L | JOHNSON |
| T3 | 123SMITH  CR | L | 123 |
| T3 | 848JOHNSONDB | L | 848 |
| T2 | 123SMITH  CR | R | CR |
| T2 | 848JOHNSONDB | R | DB |

(* The bb represents two blanks)

## 2.4.6 DATE FORMAT CODE (D)

Dates are stored in an internal format. 'D' (Date Conversion) code converts dates to or from this format. Dates may be stored in items as numbers and printed in different formats. This allows you to store dates with fewer bytes of disk space and to do mathematical calculations with stored dates. The internal format of any date is the integer number of days between that date and the zero date, December 31, 1967. Dates before 12/31/67 are stored as negative numbers; dates after 12/31/67 as positive numbers. For example:

| INTERNAL FORMAT | EXTERNAL (LISTING) FORMAT |
|---|---|
| -100 | 22 SEP 1967 |
| -10 | 21 DEC 1967 |
| -1 | 30 DEC 1967 |
| 0 | 31 DEC 1967 |
| 1 | 01 JAN 1968 |
| 10 | 10 JAN 1968 |
| 100 | 09 APR 1968 |
| 1000 | 26 SEP 1970 |
| 10000 | 18 MAY 1995 |

D codes are entered in attribute 7 of the dictionary which describes the attribute of the items which store the date in internal format. Different external formats which may be specified for the 'D' code are given below.

| D CODE | INTERNAL FORMAT | EXTERNAL (LISTING) FORMAT | |
|---|---|---|---|
| D | 5744 | 22 SEP 1983 | |
| D/ | 5744 | 09/22/1983 | |
| D- | 5744 | 09-22-1983 | |
| DO | 5744 | 22 SEP | |
| DO/ | 5744 | 09/22 | |
| D2* | 5744 | 09*22*83 | |
| D%1 | DUE%5744 | DUE%22 SEP 1983 | |
| D%1/ | BAL%5744 | BAL%09/22/1983 | |
| D#1- | CHG#5744 | CHG#09-22-1983 | |
| DO$1 | SALE$5744 | SALE$22 SEP | |
| DO | SALE$5744 | SALE$5744 | |
| D4- | 5744 | 09-22-1983 | |
| DY | 5744 | 1983 | (4-digit year) |
| D2Y | 5744 | 83 | (2-digit year) |
| DQ | 5744 | 3 | (3rd quarter) |
| DD | 5744 | 22 | (22nd day) |
| DM | 5744 | 9 | (9th month) |
| DMA | 5744 | SEPTEMBER | |
| DJ | 5744 | 265 | (265th day of year) |
| DW | 5744 | 4 | (4th day of week) |
| DWA | 5744 | THURSDAY | |
| DI | 9/22/83 | 5744 | (Reverse conversion) |
| DI]D2 | 9/22/83 | 22 SEP 83 | (Multivalue conversion) |

The 'D' (Date conversion) code has the following general form:

```
D{n}{*m}{s}
```

where:

- D is the code name
- n is an optional single digit number which specifies the number of digits to occur in the year on output. If 'n' is 0, no year will appear in the date. n = 0, 1, 2, 3, or 4 is valid; if 'n' is omitted, n = 4 is assumed.
- * stands for any single non-numeric character which specifies the delimiter between fields for group extraction. (* may not be a system delimiter.)
- m is a single digit number that must accompany * (if * is specified). m specifies the number of fields to skip for group extraction.
- s stands for either any single non-alphanumeric character (such as ' ' or '-') that may be specified to separate the day, month and year on output, or a special subcode, either D, M, Q or Y. If 's' is specified as a separator character, the date will be in the format 12-31-1982. If 's' is not specified (and a subcode is not used), the date will be in the format 31 DEC 1982. If a subcode is specified, then only the number of the day, month, quarter or year is displayed. Some examples of the use of the subcode:

| D Subcode | Display |
|---|---|
| DD | day of month |
| DM | month of year |
| DQ | quarter of year |
| D{n}Y | n-digit year (n = 1 - 4) |
| DY | 4-digit year |

Note the use of the '*m' option which will perform a Group Extration before applying the date conversion. (See Section 2.4.1, GROUP EXTRACTION CODE G.)

A special note for using the 'D' (Date conversion) code on input: if no year is specified, the current year will be used. If only two digits are specified for the year, then years entered as 30 to 99 will be stored as 1930 through 1999, and years entered as 00 to 29 will be stored as 2000 through 2029.

## 2.4.6.1 Time Formats

Time is stored in an internal format that is suitable for arithmetic processing. The MT code provides the facility for converting time to or from this compact internal format.

The internal time format is the number of seconds from midnight. The external time is 24-hour military format (e.g., 23:25:59) or 12 hour format (e.g., 11:25:59PM). The general form of the MT code is as follows:

```
MT{H}{S}
```

where:

| | |
|---|---|
| MT | is the Mask Time code specification |
| H | is the capital letter H which optionally specifies 12 hour format. If 'H' is omitted, 24 hour (military) format is assumed. |
| S | is the capital letter S which optionally specifies seconds on output. If 'S' is omitted, seconds are not listed on output. |

When codes MTH and MTHS are used, 12 hour external format is specified. For input conversion, then, the time is entered with AM or PM immediately following the numeric time (AM is optional). On output, AM or PM is always printed immediately following the numeric time.

NOTE: 12:00 AM is considered midnight, and 12:00 PM is considered noon. AM and PM will be ignored on input if code MT is specified. Illegal values are converted to null on input. Sample usage of MT:

| MT CODE | INPUT VALUE | STORED VALUE | OUTPUT VALUE |
|---|---|---|---|
| MT | 12: | 43200 | 12:00 |
| MTH | 12PM | 43200 | 12:00PM |
| MTH | 12AM | 0 | 12:00AM |
| MTS | 12AM | 0 | 00:00:00 |
| MTHS | 12AM | 0 | 12:00:00AM |
| MT | 12:15AM | 900 | 00:15 |
| MTH | 12:15AM | 900 | 12:15AM |
| MT | 1AM | 3600 | 01:00 |
| MTH | 1AM | 3600 | 01:00AM |
| MT | 6AM | 21600 | 06:00 |
| MTH | 6AM | 21600 | 06:00AM |
| MT | 1PM | 46800 | 13:00 |
| MTH | 1PM | 46800 | 01:00PM |
| MT | 13: | 46800 | 13:00 |
| MTH | 13: | 46800 | 01:00PM |
| MT | XYZ | 0 | 00:00 |

## 2.4.7 FILE TRANSLATION CODE (Tfile)

The Tfile code provides a facility for converting a value by translating through a file. The value to be translated is used as an item-id for retrieving an item from the defined translation file. The translation value is retrieved from the specified attribute of the item. The general form of the Tfile (Translate) code is:

```
    T{DICT}file;cn;i-amc;o-amc{;b-amc}
```

where:

| | |
|---|---|
| T | is the code name. |
| file | is the name of the file through which translation takes place, The file name may be preceded by "DICT" to indicate a dictionary. |
| c | is the translate subcode, which must be one of the following: |

| | |
|---|---|
| V | Conversion item must exist on file, and the specified attribute must have a value. Aborts with an error message if translation is impossible. |
| C | Convert if possible; use original value if item in translate file does not exist or has null conversion attribute. |
| I | Input verify only (functions like 'V' for input and like 'C' for output). |
| O | Output verify only (functions like 'C' for input and like 'V' for output). |
| X | Convert if possible; otherwise return a null value. |

| | |
|---|---|
| n | is an optional value mark count specification. If the c element is followed by a number, the translate will return only the value in VMC n, instead of the complete collection of values concatenated together with blanks. Subvalues will be returned with included blanks. |
| i-amc | is the decimal attribute number for input conversion (in BASIC or BATCH). The input value is used as an item-id in the specified file, and the translated value is retrieved from the attribute specified by the i-amc. (If the i-amc is omitted, no input translation takes place.) |
| o-amc | is the attribute mark count for output translation. When ACCESS creates a listing, the attribute values will be looked up in the specified file, and the attribute specified by the o-amc will be listed instead of the original value. |
| b-amc | if specified, will be used instead of o-amc during the listing of break-on and total lines. |

Sample usage of the Tfile (Translate) code:

Item CARDS in DICT MUNCH is:

```
001  S
002  4
003  CREDIT CARDS
004
005
006
007  TCARD-FILE;C;;1
008
009  L
010  20
```

Data section of MUNCH file is:

```
Item-id:   COCOS             CARLS-JR          MEYERHOFS
    001:   HAMBURGERS        HAMBURGERS        SANDWICHES
    002:   MAC ARTHUR BLVD.  BRISTOL STREET    S.COAST VILLAGE
    003:   5583233           9792231           8830002
    004:   MC]V              NONE              MC]V]BA
```

Data section of CARD-FILE is:

```
Item-id:   MC                V                 BA
    001:   MASTER CHARGE     VISA              BANKAMERICARD
```

```
>LIST MUNCH "COCOS""CARLS-JR""MEYERHOFS" CARDS HDR-SUPP  [CR]

COCOS        MASTER CHARGE
             VISA
CARLS-JR     NONE
MEYERHOFS    MASTER CHARGE
             VISA
             BANKAMERICARD
```

## 2.4.8 ASCII AND USER CONVERSION CODES (MX AND U)

The MX code is used to convert strings to or from their hexadecimal equivalents. The 'U' code allows the user to write his own conversions in assembly language.

The MX code specifies that character strings are to be converted, one character (byte) at a time, into their hexadecimal (base sixteen) representations. Each character will be converted to a 2-digit (one byte) hexadecimal number. This feature is useful in finding non-printable characters in data strings. The general form of the MX (Mask Hexadecimal) code is:

MX

The mask decimal codes may be combined with the mask character code as shown:

| Code | Action |
|---|---|
| MCDX | Converts decimal data to hexadecimal equivalent. |
| MCXD | Converts hexadecimal data to decimal equivalent. |

Sample usage of the MX (Mask Hexadecimal) code:

| INPUT VALUE | CONVERTED VALUE |
|---|---|
| ABC | 414243 |
| JOHN | 4A4F484E |
| john | 6A6F686E |
| HI THERE | 4849205448455245 |

The 'U' code specifies an entry point into a user-written piece of software. the general form of the 'U' (User-defined) code is:

Unxxx

where:

| | |
|---|---|
| U | is the code name |
| n | is the entry point number, and |
| xxx | is the hexadecimal FID (Frame ID) of the frame containing the user's assembly codes. |

WARNING: Do not use U code unless you fully understand its action at the assembly-code level.

## 2.4.9 MATHEMATICAL OR STRING FUNCTIONS CODE (F)

The 'F' code is used to perform mathematical operations on the attribute values of an item, or to manipulate strings.

All operations specified by an 'F' code operate on the last two entries in a pushdown stack. This pushdown stack may be visualized as follows:

```
          ------------
STACK1    :          :
          ------------
STACK2    :          :
          ------------
STACK3    :          :
          ------------
STACK4    :          :
          ------------
STACK5    :          :
          ------------
etc.
```

STACK1 is the top position in the stack, STACK2 is the next position, etc. As a value is pushed onto the stack, it is pushed into position STACK1; the original value of STACK1 is pushed down to STACK2; and so on.

An 'F' code is comprised of any number of operands or operators in reverse Polish format separated by semicolons. When an operand specification (a numeric attribute number or constant, or a string) is encountered, the value is "pushed" onto the top of the stack. When an operator is encountered, the specified operation is carried out on the top one or two entries in the stack, depending upon the operator. When the entire 'F' code has been processed, the value printed is the value in the top of the stack.

The general form of the 'F' code is as follows:

```
Felement;element;element...
```

An "element" may be any of the following: a numeric AMC specifying an attribute value to be pushed onto the stack (optionally followed by an "R" to specify that the first value or subvalue of an attribute is to be used repeatedly when using it against a multivalued value or subvalue).

An element may also be of the form 'Cn' where 'n' is a numeric constant to be pushed onto the stack; a 'D' which specifies that the current date is to be pushed onto the stack; a 'T' which specifies that the current time is to be pushed onto the stack; a special 2-character operand; or an operator which specifies an operation to be performed on the top two entries in the stack. The operators are listed in the following table.

| Operator | Operation |
|---|---|
| *{n} | Multiplication of the top two entries in the stack. If the optional "n" is used, the result is "descaled" by n, that is, it is divided by 10**n. |
| / | Division of STACK2 by STACK1, result to STACK1. |
| R | Same as "/" but remainder is returned to top of stack (instead of quotient). |
| + | Addition of the top two entries in the stack, result to STACK1. |
| - | Subtraction of STACK1 from STACK2, result to STACK1. |
| : | Concatenate; the string value from STACK1 is concatenated onto the end of the string value from STACK2. |
| [] | Substring; a subset of the string value from STACK3 is extracted, using STACK2 as the starting character position, and STACK1 as the number of characters to extract; the result is placed on top of the stack. This is equivalent to the BASIC [m,n] operator, where "m" is in STACK2 and ´n´ in STACK1. |
| S | A total sum of all previous computations is placed at the top of the stack. |
| _ | Underline. Exchanges top two positions in stack. |
| P | Pushes the top stack value back onto the stack; that is, it duplicates the top stack value. |
| (...) | Conversion operator; a standard conversion operator such as D (date), G (group), etc. may be specified and will operate on the top stack value; the result will replace the original top stack value. |

The following operators operate on the top 2 stack entries, and a result of zero or one is placed on the top of the stack, depending on whether the condition is not or is satisfied.

| | |
|---|---|
| = | "Equal" relational operator. |
| > | "Greater than" relational operator. |
| < | "Less than" relational operator. |
| # | "Not equal" relational operator. |
| ] | "Equal to or greater than" relational operator. |
| [ | "Equal to or less than" relational operator. |

The relational operators compare STACK1 to STACK2; after the operation, STACK1 will contain either a 1 or 0, depending upon whether the result is true or false, respectively (e.g., if the 'F' code were F;C3;C3;= then STACK 1 would contain a 1). Examples of F are:

F2;3;4;*;+ is equivalent to:

(attribute3 * attribute4) + attribute2

F23;24;*;C100;+;P;C0;<;* is equivalent to:

(attribute23 * attribute24) + 100 ;
If this result is < zero, final result is zero; else the above value is returned as the result. (The above value is generated, and is repeated in the stack via the P operator; it is then compared to zero, which gives a result of 0 or 1 depending on whether it was less than zero or not; this is muliplied by the original value, giving a zero or the original value.)

F3;4;*;6R;+;S is equivalent to:

(attribute3 * attribute4) + attribute6
If attributes 3 and 4 are multivalued, and 6 is not, the single value in attribute 6 will be used repeatedly in the addition (if the "R" is not present, it will be used only once, and zeros will be used for other computations); a sum of such computations for all multivalues in attribute3 or attribute4 (whichever has the greater number of multivalues) is returned.

F3;" ";:;4;:;5;C1;C10;[];: is equivalent to:

attribute3:" ":attribute4:attribute5[1,10]
That is, the value from attribute 3 is concatenated to that from attribute 4, with a space between them; the first through the 10th characters from attribute 5 are then concatenated to the end of that result.

### 2.4.9.1 Multivalued and Special Operands (Code F)

The 'F' code operands may be multivalued, may contain conversion specifications, or may be a special 2-character operand specifying one of several counters. Different interpretations are given to an 'F' correlative versus an 'F' conversion for an attribute with a TOTAL modifier.

When arithmetic operations are performed on two multivalued lists (vectors), the answer will also be multivalued and will have as many values as the longer of the two lists. Zeros will be substituted for the null values in the shorter list. For example, suppose attribute 10 had values of "5]10]15" and attribute 15 had values of "20]30]40]50"; if the correlative F;10;15;+ were processed, the result in STACK1 would be "25]40]55]50". If a single valued attribute is to be repetitively added (or subtracted, etc.) with a multivalued attribute, then the single letter R should immediately follow the AMC in the 'F' code (e.g., F;10;25R;+).

Any conversion may be specified in the body of a Function correlative. The conversion specification must be enclosed by parentheses. Examples of F correlation with conversions:

```
F;10;11;(TDICT SALES;X;3;3);*
```

Places the data from attribute 10 in the stack; translates the data from attribute 11 through the dictionary of file-name SALES and stacks it; then multiplies the two numbers together.

```
F;D;(DY);3;(DY);-
```

Computes the difference in years between the current date and the date in attribute 3.

```
F;1;(ML#10);2;:
```

Concatenates the data in attribute 2 with the result of applying the format string "L#10" to attribute 1.

Special 2-character operands may be used as 'F' code elements as listed below:

| Operand | Description |
|---|---|
| NI | Current item counter (number of items listed or selected). |
| ND | Number of Detail lines since last BREAK on a Break line. On a detail line it has a value of 1. On a grand-total line, it equals the item counter. (Used to generate averages in conjunction with control breaks.) |
| NV | Current multivalue counter for columnar listing only. |
| NS | Current submultivalue counter for columnar listing only. |
| NB | Current Break level number; 1 = lowest level break; this has a value of 255 on the grand-total line and a value of zero at detail time. The lowest level control-break, the one on the right in the sentence, will have a value of 1. |
| LPV | Will load the result of the last conversion onto the stack. |

For example:

```
F;ND;3;/
```

On every detail line, this returns the value from attribute 3; on every Break line (including the grand-total line), the average value of the data in attribute 3 is returned. (This must be specified as a conversion in line 7.)

The Function code operates in two different fashions on an attribute with a TOTAL modifier, depending on whether it is specified as a correlative or as a conversion. As a correlative, the function is applied before the accumulation of the total, and is ignored on the Break data line; therefore, the total of the functioned value is computed. As a conversion, the function is ignored on detail lines, and is applied only on the Break data line and other subtotal fields in the output. Therefore, the function of other totaled values is obtained. If the function is specified as a conversion, the numeric operand (AMC) in the Function code must correspond to an attribute that is being totaled within the statement. If such an attribute does not exist, a value of zero is returned. Note that the numeric operators may be dummy AMCs in that they may reference other attributes within the statement that have function correlatives.

## 2.4.9.2 The Load Previous Value (LPV) Operator

The function processor commences operation with no prior data. If attribute 8 commences with an F-correlative, there is no prior data set up by any processor in the system. Entering attribute 7 there is the result of attribute 8 or at least the data retrieved from the item according to the specification in attribute 2 of the data definition item. It is possible to load this data into the function correlative stack using the LPV instruction. Noting that a conversion may call a function correlative, we may also load the last result of a series of conversions within a given attribute definition line into a function which follows the conversion in the line. For instance:

```
    DATA DEFINITION ITEM
001 A                               Data definition item mark.
002 3                               Specifies data attribute 3.
    .
    .
007 F;LPV;"100";/                   Will divide the result of
                                    attribute 8 by 100.
008 F;2;3;*                         Contents of data attribute 2
                                    times the contents of data
                                    attribute 3.
```

If this data definition item is totaled, the total generated will be loaded into attribute 7 and divided by 100 prior to output on the break line.

```
002 5                               Data attribute 5.
    .
    .
008 G*1]MRZ8]F;LPV;"52";R;"*C";:]TFILE;C;;3
```

This has rather less motivation, since it is equivalent to:

```
008 F;5(G*1]MRZ8);"52";R;"*C";:;(TFILE;C;;3)
```

The LPV should not be used as the first operator in a F-correlative, because it has the effect of loading the contents of the temporary data area into the stack. If the LPV is used at other points in an F-correlative, strange things will happen.

The LPV operator is available for use in A-correlatives.

### 2.4.9.3 Summary of Code F Stack Operations

The following operands are pushed onto the stack, and all other stack elements are pushed down one level.

| Operand | Action |
|---|---|
| n{R} | A decimal number. The attribute value for the corresponding attribute number is stacked. The optional R specifies that the value is to be repeated if it is a single value and there are multiple values in other attributes which are part of the stack operation. |
| Cn | (where ´n´ is a decimal number) The integer constant ´n´ is stacked. |
| "String" | The literal string enclosed in double quotes is stacked. |
| ´String´ | The literal string enclosed in single quotes is stacked. |
| D | The current date (in internal format) is stacked. |
| T | The current time (in internal format) is stacked. |
| P | The top stack element is pushed back onto the stack. |
| NI | The current item counter is stacked. |
| ND | The number of detail lines since the last control break is stacked. (This number is 1 on detail lines, and is the same as the item counter on grand-total lines.) |
| NS | The current submultivalue counter is stacked. |
| NB | The current control-break number is stacked. |

The following operator pops three entries off the stack, computes a result, and pushes it onto the top of the stack.

| | |
|---|---|
| [] | The substring from the string value in the third stack element is extracted, starting from the character position defined in the second stack element, and the number of characters to extract is defined in the first stack element. For example, the operation F4;C5;C8;[] extracts the string in attribute 4 and places its fifth through twelfth characters on the top of the stack. |

The following operators pop the top two operands from the stack, compute a result, and push the result onto the stack. All other stack elements are popped up one level.

| | |
|---|---|
| + | The top two elements are added together and the sum is stacked. |
| - | The first stack element is subtracted from the second and the difference is stacked. |
| *{n} | The top two elements are multiplied and the result is stacked. If n is specified, the result is divided by 10**n before it is stacked. |
| / | The second stack element is divided by the top stack element and the dividend is stacked. |
| R | The second stack element is divided by the top stack element and the remainder is stacked. |
| : | The top stack element is concatenated onto the end of the second stack element and the resultant string is stacked. |

The following relational operators compare the two top elements of the stack, pop them both off the stack and then push a 1 (TRUE) or 0 (FALSE) onto the stack.

| | |
|---|---|
| = | The top two elements of the stack are compared, a 1 is stacked if they are equal, a 0 is stacked if they are not equal. |
| # | Stacks a 1 if the top two stack elements are unequal; stacks a 0 if they are equal. |
| > | Stacks a 1 if the top stack element is greater than the second stack element, stacks 0 otherwise. |
| < | Stacks a 1 if the top stack element is less than the second, a 0 otherwise. |
| ] | Stacks 1 if the top element is greater than or equal to the second element, 0 otherwise. |
| [ | Stacks 1 if the top element is less than or equal to the second element, 0 otherwise. |

The following operators function on just the top one or two stack entries and have no effect on the rest of the stack.

| | |
|---|---|
| S | Sums the multiple values (if any) of the top stack element. |
| _ | Underline. Exchanges the first and second stack elements. |

## 2.4.10 MATHEMATICAL FUNCTION CORRELATIVE (A)

The A-code is designed to perform the same function as the F-code, but its format is both simpler to write and easier to understand than the format of the F-code. The general form of the A-code is:

    A(expression)

where an expression is made up of operands, functions and operators. Like any mathematical expression, parentheses may be used to indicate the precedence of the operations.

### 2.4.10.1 Operands

1. AMC Numbers - An Attribute Mark Count (AMC) is specified by putting the number in the A-code, just like in the F-code. An AMC of 0 (zero) will indicate the Item-Id. The special AMCs 9999 and 9998 retain their original functions and can be legally inserted into an A-code. An AMC can optionally be followed by the letter "R", which indicates repetition of the value just like in F-codes.

2. AMC Names - An attribute name can be used instead of an AMC in an A-code as long as the name exists in the dictionary of the file being listed. The dictionary name is used as an argument to the "N" function of the A-code. The format of the "N" function is N(attribute-name). For example, if the name INVOICE-AMOUNT exists in the dictionary, the corresponding "N" function would be N(INVOICE-AMOUNT).

The operation of the "N" function is:

The attribute-name is referenced in the dictionary of the file and an error message is printed if it is not found. The AMC of the dictionary item (attribute 2) is used as the AMC in the A-code.

Any correlatives existing in attribute 8 of the dictionary item, whether F-codes or A-codes, will be used in the A-code. If an A-code or F-code exists in attribute 8 of the dictionary item, the AMC from attribute 2 is ignored.

Note that an A-code can call another A-code by name and that the second A-code can specify a third A-code, and so on. However, no attempt is made to assure that an A-code does not call itself. If this is attempted, or any time that "nested" calls are made more than seven levels deep, the ACCESS compiler will abort with a RTN STACK FULL message.

3. Literal Numbers - A number is specified by enclosing the number in quotes, either single (') or double ("). For example, the number 10 could be specified by "10" or '10'. Any integer, positive, negative or zero is legal inside quotes.

4. Literal Strings - Any literal string, enclosed in single quotes (´) or double quotes (") is a legal operand.

5. Special Operands - The A-code has several special system operands which are the same as the ones for F-codes.

| | |
|---|---|
| NI | is the item counter |
| NV | is the value counter |
| NS | is the subvalue counter |
| ND | is the detail line counter |
| NB | is the break level counter |
| LPV | is the load previous value |
| D | is the system date (in internal format) |
| T | is the system time (in internal format) |

These special operands can be used exactly like an AMC, attribute name or literal. Also, any of the above legal operands preceded by a minus sign, (-) is a legal operand.

2.4.10.2 Functions

1. Remainder Function: "R" - The remainder function takes two expressions as operands and returns the remainder of the first operand divided by the second. The format of the "R" function is R(expression,expression). For example, R(2,"5") returns the remainder when the value of attribute 2 is divided by 5.

2. Summation Function: "S" - The summation function takes one expression as an operand and works the same way as the S operator in the F-codes. For example, S(4) will sum any multivalues of attribute 4. The summation operator may appear anywhere in an A-code. A maximum of 2 summations is allowed.

3. Substring Function - A subtring may be specified by using square brackets. The numbers inside the brackets may be literal numbers, AMCs, or entire expressions. For example, 1["2",´3´] means the 3-character long string starting at position 2 of attribute 1. The expression 1["1",´99´*(2=4)] will evaluate to the value of attribute one, unless attributes two and four are different, in which case the expression evaluates to a null string.

### 2.4.10.3 Operators

1. Arithmetic Operators - The operators +, -, * and/ denote addition, subtraction, multiplication and division, respectively. All of the arithmetic operators take two expressions as operands, and return the sum, difference, product or quotient of the two operands. It is important to note that division in an A-code always returns an integer result, just like in F-codes, so that "3"/"2" evaluates to 1, not 1.5.

2. Relational Operators - The relational operators >, <, >=, <=, =, and # denote the logical relations greater than, less than, greater than or equal to, less than or equal to, equal and not equal, respectively. Each of the relational operators takes two expressions as operands, and evaluates to 1 (TRUE) or 0 (FALSE) depending whether or not the indicated relation holds between the two operands. For instance, "1">="2" evaluates to 0 (FALSE) because the number 1 is not greater than or equal to the number 2. Expressions involving these and other A-code operators are written much like BASIC expressions.

NOTE: The precedence of the operators is important to keep in mind when writing an A-code and not using parentheses to indicate the order in which operators are to be applied. Multiplication and division have greater precedence than addition and subtraction, which in turn have greater precedence than the relational operators. If two operators have the same precedence, they are applied from left to right. For example, 1*2+3<4 will evaluate as ((1*2)+3)<4, but 1>=2-3/4 will evaluate as 1>=(2-(3/4)). Also, 1+2-3 will evaluate as (1+2)-3, and 4/5*6 will evaluate as (4/5)*6. 20 levels of parentheses nesting are allowed in A-codes.

A-Code operator examples:

| A-Code | Meaning |
|---|---|
| A1+2 | Adds attributes 1 and 2. |
| A"10"*3 | Multiplies the value of attribute 3 by 10. |
| AS(4+"25") | Adds 25 to each value of attribute 4, then sums the multivalues. |
| AN(INV-AMT)-N(BAL.DUE) | Subtracts the value of the attribute defined by BAL.DUE in the dictionary from the value of the attribute defined by INV-AMT. |
| AN(SS-NUM)['4','2'] | Returns the 4th and 5th digits of the attribute specified by SS-NUM. |

## 2.4.11 PROCESSING AND FORMATTING NUMBERS - CODES (MR AND ML)

The MR and ML codes allow special processing and formatting for numbers or dollar amounts. These codes should be used in the conversion attribute (attribute 7) of the item dictionary for best results.

The ACCESS MR and ML codes function exactly like the 'R' and 'L' format strings in BASIC. The general form of the MR and ML codes is:

```
    M(R/L){n{m}}{Z}{,}{C/D/M/E/N}{$}{(format-mask)}
```

where:

| | |
|---|---|
| M | is the code name. MR specifies the number to be right justified; ML specifies left justification. Note that the L or R justification specification in attribute 9 of the dictionary item will override the MR or ML except when a format-mask is also used. |
| n | is a single decimal digit (0-9) which specifies the number of digits to be printed to the right of the decimal point. If 'n' is not specified, 0 is assumed. If 0 is assumed or specified, no decimal point will be printed. |
| m | is a single digit number (0-9) which specifies that the number on file is to be 'descaled' (divided) by that power of ten. (That is, if m=2, the number is divided by 100, if m=3, the number is divided by 1000, and so on.) The number 'm' is the number of implied digits to the right of the decimal point for the number as it is stored in the file. If m>n, then the number will be rounded off, either up or down, to 'n' digits. |
| Z | is the optional zero-suppress parameter. If 'Z' is specified, the 0 (zero) numbers will be printed as blanks. |
| , | specifies insertion of commas every three digits to the left of the decimal point. |
| C | causes negative values to be followed by the letters CR. |
| D | causes positive values to be followed by the letters DB. |
| M | causes negative numbers to be followed by a minus sign (-). |
| E | causes negative numbers to be enclosed inside angle brackets (< and >). |
| N | causes the minus sign on negative numbers to be suppressed. |
| $ | appends a dollar sign ($) to the number before justification. |

The format mask specification, which is enclosed in parentheses, consists of format codes and literal data. A format code is one of the characters #, *, or %, optionally followed by a number to indicate that number of repetitions of the characters.

| Format Mask | |
|---|---|
| (#n) | specifies data to be justified in a field of 'n' blanks. |
| (*n) | specifies data to be justified in a field of 'n' asterisks (*). |
| (%n) | specifies justification in a field of 'n' zeros (0). |

Any amount of alphabetic data may also be specified inside the parentheses in the format mask specification. The data will be printed exactly as specified in the format mask, with the number being processed appearing either right or left justified in the place of the #'s, *'s, or %'s. Any #'s, *'s, or %'s not overwritten by the number will appear either to the left or right of the number as specified.

Note the $ option apends a dollar sign to the number and then justifies the number, so the dollar sign will always appear just before the first digit of the number on output.

Sample usage of the MR and ML codes:

```
Data            Conversion        Result

1234            MR2               12.34
1234            MR                1234
1234            MR(%10)           0000001234
1234            MR(*10)           ******1234

12345678        MR2,E             123,456.78
-12345678       MR2,E             <123,456.78>
-12345678       MR2,C$            $123,456.78CR

572082394       MR24              57208.24 (Note rounding)
572082394       MR2,$(#20)               $5,720,823.94
572082394       MR2,($#20)        $        5,720,823.94
572082394       MR2,$(*20)        *******$5,720,823.94

572082394       ML(###-##-####)   572-08-2394
572082394       MR(#3-#2-#4)      572-08-2394
572082394       ML(#3-#4 EXT.#2)  572-0823 EXT.94
```

## 2.4.12 MASK CHARACTER CODE (MC)

The MC (Mask Character) code allows the user to change attribute data to upper or lowercase or to select certain classes of characters.

The following is a list of the forms of the MC code and their function:

| | |
|---|---|
| MCU | Converts data to uppercase. Will change all lowercase letters to uppercase; has no effect on uppercase letters or non-alphabetic characters. |
| MCL | Converts data to lowercase. Will change all uppercase letters to lowercase; has no effect on lowercase letters or non-alphabetic characters. |
| MCT | Converts uppercase characters to lowercase, starting with the second letter in each word. |
| MCP | Converts all nonprintable characters to periods (i.e., X'00'-X'1F'). |
| MCA | Extracts and prints all alphabetic characters, either uppercase or lowercase; non-alphabetic characters will be deleted from the data. |
| MCN | Extracts all numeric characters (0-9) from the data; deletes all other characters. |
| MC/A | Extracts all non-alphabetic characters from the data; deletes all alphabetic characters. |
| MC/N | Extracts all non-numeric characters; deletes all numeric characters. |

Sample usage of the MC (Mask Character) code:

| Value | MC Code | Output Value |
|---|---|---|
| JOHN SMITH 1234 | MCL | john smith 1234 |
| John Smith 1234 | MCU | JOHN SMITH 1234 |
| John Smith 1234 | MCA | JohnSmith |
| John Smith 1234 | MCN | 1234 |
| 572-08-2394 | MCN | 572082394 |
| (714) 552-4275 | MCN | 7145524275 |
| abc123$%XYZ | MC/A | 123$% |
| abc123$%XYZ | MC/N | abc$%XYZ |

# advanced access 3

## 3.1 USING THE SELECTION PROCESSOR

This section gives an advanced description of the operation of the selection processor. You should be familiar with Section 1.19, Forming Selection-Criteria, before reading this section.

### 3.1.1 ITEM-ID SELECTION

ACCESS allows you to select item-ids in different ways. The default is to select the whole file, unless item-ids were previously specified in a list, and that list is active. Otherwise, item-ids may be selected by putting restrictions on them in two ways which are syntactically similar, but which operate differently.

#### 3.1.1.1 Explicit Item-ids

If explicit item-ids are specified, then only those item-ids will be returned. This will take precedence over any list in effect, which will be ignored.

Explicit item-ids are specified as follows:

```
    LIST FILENAME 'ITEM1''ITEM2''ITEM3'
```

or

```
    LIST FILENAME "ITEM1""ITEM2""ITEM3"
```

or even

```
    LIST FILENAME \ITEM1\\ITEM2\\ITEM3\
```

Each sentence will yield a listing of the three items, ITEM1, ITEM2, and ITEM3. The processor will retrieve each of these items direcly from the file referenced. This collection of explicit item-ids then becomes a list which the processor uses to obtain successive item-ids until the list is exhausted, at which time the process terminates.

The delimiter ' is reserved for item-id specification. The delimiters " and space are normally used for value specification, but will be taken as item-ids under certain circumstances.

## 3.1.2 ITEM-ID TESTS

ACCESS allows tests to be used on item-ids as a method of selecting them. Be aware that all tests are on the item-id as it stands in the file. No conversions or correlatives will be applied to the item-id before the test is made.

To specify an item-id test rather than the retrieval of a specified set of items you include a relational operator, hereinafter referred to as a relational connective, in the item-id specification string. For example:

```
    LIST FILENAME 'ITEM1' = 'ITEM2''ITEM3'
```

or

```
    LIST FILENAME "ITEM1""ITEM2" = "ITEM3"
```

or even

```
    LIST FILENAME = \ITEM1\\ITEM2\\ITEM3\
```

will all have the same effect. ACCESS will search the whole file or all the items in a list, if there is one in effect, looking for items which have the item-ids ITEM1, ITEM2, or ITEM3. (An active list will take precedence over the file.) This will take longer than using the simpler explicit item-id reference, and is not recommended when you know which item-ids you want. Note that in these examples, only one relational connective was included in each sentence. All the elements not preceded by a connective are automatically assigned the connective '='. This is true throughout ACCESS in cases when values are used.

Relational connectives are used to allow specifications of the form

```
    LIST FILENAME < 'CAT'
```

which will have the effect of selecting all items which are lexicographically less than CAT, presuming that the file is left-justified. The full effect of justification will be considered later.

Spaces between the item, or value strings, and the file name are optional. Data definition items or connectives may be concatenated to a value in the input sentence in the form:

```
    ="ITEM1"
```

In all other cases, all elements in an ACCESS sentence which are to be retrieved from a dictionary or dictionary-equivalent file must be surrounded by blanks.

You may either specify an explicit list of item-ids for retrieval or a test on item-ids. It is not possible to retrieve certain items directly and test other items for admissability using item-id tests. In other words, the item-id list is either a list of explicit item-ids or a sequence of values against which to test each item-id in the file. The difference is the inclusion of a relational connective in the item-id list.

## 3.1.3 THE USE OF DELIMITERS IN THE ACCESS SENTENCE

In previous examples, the delimiters ′, ", and \ were used equivalently to delimit item-ids or values. In general, only the delimiter ′ is reserved for item-ids or item-id-related values. The " and \ may be used for values related to data definition item selection criteria and print-limiters as well. It is preferable to use them with data selection-criteria rather than with item-id selection-criteria. In general, values delimited by either " or \ will be treated as item-id selection-criteria only if they follow the file reference and precede a data definition item.

The delimiter ′ will cause the value which it surrounds to be treated as an item-id selection criterion wherever that value may be in the sentence.

## 3.1.4 ITEM-ID SELECTION CRITERIA

Presume that we have a set of values with an associated relational connective, so that ACCESS is scanning the whole file in order to test the item-ids for acceptability. Be aware that the item-id test is logically ANDed with all other selection criteria. If an item-id fails the item-id test, the item will be discarded. If one wishes to ′OR′ an item-id test with other selection criteria, then a data definition item must be included which references the item-id. For an example, see the next page.

Consider the following dictionary attribute items.

|  |  |  |
|---|---|---|
| 0 | 1 | item-id |
| 001 A | 001 A | typifier |
| 002 0 | 002 1 | AMC specifier |
| 003 | 003 | Null label |
| 004 | 004 |  |
| 005 | 005 |  |
| 006 | 006 |  |
| 007 | 007 |  |
| 008 | 008 |  |
| 009 L | 009 L | Justification |
| 010 10 | 010 10 | Length |

and the following ACCESS sentences:

|  |  |
|---|---|
| LIST MD < "CAT" AND WITH 1 = "D" | This will select all items whose item-ids are characters whose sorting evaluation is less than 'CAT' AND which have a 'D' in attribute 1. |
| LIST MD WITH 0 < "CAT" WITH 1 = "D" | This will select all items whose item-ids are evaluated less than 'CAT' OR which have a 'D' in attribute 1. Note that by referring to the item-id as attribute 0, the item-id is considered like any other attribute. |
| LIST MD WITH 0 < "CAT" AND WITH 1 = "D" | This will select all items whose item-ids are evaluated less than 'CAT' AND which have a 'D' in attribute 1, as in the first case. |
| LIST MD WITH 0 < "CAT" OR 1 = "D" | This is erroneous. It will have the effect of selecting all items whose item-ids satisfy the CAT criterion only. The rest of the sentence will be ignored. |
| LIST MD < "CAT" AND WITH 1 = "D" | This is erroneous. It will terminate in the ACCESS compiler with an error, because the AND connective must be followed by another value which may be ANDed with CAT, and because the AND connective may not immediately precede the first WITH connective in the ACCESS sentence. |

### 3.1.5 SELECTION BY DATA VALUE

Attributes may be used as selection-criteria in the ACCESS sentence by preceding the attribute-name with the connective WITH. The standard form of a WITH phrase is

```
    WITH {NOT} {EACH} attribute-name {"value-list"}
```

and is called a selection-criterion. For convenience, the connectives NOT and EACH following the WITH are referred to as selection modifiers. The basic form of selection

```
    WITH attribute-name
```

will select an item if it contains any value for the named attribute other than null. On the other hand,

```
    WITH NOT attribute-name  or  WITHOUT attribute-name
```

will select an item only if the attribute named contains null. The modified form:

```
    WITH EACH attribute-name
```

will only succeed if each value in the named attribute is non-null. However, the form:

```
    WITH NOT EACH attribute-name  or  WITHOUT EACH attribute-name
```

will accept all items that contain at least one null value returned for the named attribute.

The meaning of the term 'value' in this context will be considered after the next section.

#### 3.1.5.1 Evaluation of Data by the Selection Processor

The selection processor processes the data according to attribute 8 of the attribute definition item. That is, it executes any conversions or correlatives which are in attribute 8. The result of this calculation is returned to the selection processor. Any conversions or correlatives in attribute 7 of the data definition are not applied to data values. The contents of attribute 7, however, may have a significant effect on the success of the selection process (see Section 3.1.7).

### 3.1.5.2 Obtaining a Value to Test

The selection processor will ignore leading null subvalues within each value. That is, if an attribute of a data item contains multiple values which themselves contain subvalues, as shown below,

```
    ^ \\35\4]603\\7]1\2\30]]\\ ^
```

then the processor will retrieve one subvalue string (the first non-null) from each value. In this case, the strings returned will be:

```
    35;  603;  1;  null;  null
```

The strings returned may be original strings or strings computed by an A or F correlative. In either case, if the search for the string results in a null followed by a subvalue mark, the processor will proceed to the next subvalue in the value. It will end the data search if a non-null string has been retrieved, if it encounters a null followed by a value mark, or if it encounters a null followed by an attribute mark.

If the search results in a null data value, the process returns to the value test routine. If a non-null data value is returned, the process will then execute any conversions remaining in attribute 8 of the attribute definition item. The data resulting from the conversion, if any, will then be returned to the value test.

This is the 'value' referred to above. Note particularly that only the first non-null subvalue is returned. If another value is requested, then the next value is taken from the next value (i.e., from the right side of the next value mark if there is one). All subvalues which follow the first non-null subvalue are never inspected by the selection processor.

### 3.1.5.3 The Test for Existence

What occurs at the value test level when testing for existence depends upon the selection modifiers. If there are no modifiers, then if any non-null subvalue is returned from the item, the selection phrase will succeed. If a null value is returned, then the tester will request the next value in the item, unless the last null value was terminated by an attribute mark. In this case, the values within this attribute of this item have been exhausted and the item does not have the required value. Therefore, this selection phrase fails.

If the selection modifier is NOT, then all values defined by the attribute definition must be inspected in order that the selection phrase succeed. If any value is returned which is non-null, then the selection clause will fail.

If the selection modifier is EACH, then all values defined by the data definition must be inspected in order to succeed. If any value is returned which is null, then the clause will fail.

If the modifiers WITH NOT EACH or WITHOUT EACH are used, then the clause will succeed if any value is null. It will fail only if all values are non-null. Success and failure conditions for WITH and its modifiers are summarized below.

Success conditions for WITH and its modifiers under the test for existence:

```
    WITH attribute-name                succeeds if

    (value1 # NULL)  OR  (value2 # NULL)  OR  (value3 # NULL)  OR ...

    WITHOUT attribute-name             succeeds if

    (value1 = NULL) AND (value2 = NULL) AND value3 = NULL) AND ...

    WITH EACH attribute-name           succeeds if

    (value1 # NULL) AND (value2 # NULL) AND (value3) # NULL) AND ...

    WITHOUT EACH attribute-name        succeeds if

    (value1 = NULL) OR (value2 = NULL) OR (value3 = NULL) OR ...
```

Failure conditions for WITH and its modifiers under the test for existence.

```
    WITH attribute-name                fails if

    (value1 = NULL) AND (value2 = NULL) AND (value3 = NULL) AND ...

    WITHOUT attribute-name             fails if

    (value1 # NULL) OR (value2 # NULL) OR (value3 # NULL) OR ...

    WITH EACH attribute-name           fails if

    (value1 = NULL) OR (value2 = NULL) OR (value3 = NULL) OR ...

    WITHOUT EACH attribute-name        fails if

    (value1 # NULL) AND (value2 # NULL) AND (value3 # NULL) AND ...
```

### 3.1.6 THE VALUE STRING

In the syntax of the WITH phrase, there is an optional value-list which has not been mentioned since all of the tests for existence assume a null list as the value. A value-list is made up of value phrases of the following form:

    {relational connectives} "value"

The relational connectives are optional in the sense that the relation will default to '=' if there is no relational connective preceding the value.

A value is of the form:

    "text-string"

or

    /text-string/

(Remember that the delimiter ' specifies an item-id reference.)

The contents of the text string may be any characters with the exception of the system delimiters. Avoid the control characters, if possible. There are three special symbols, ^, [, and ] which have a special meaning to the selection processor, and will be considered in Section 3.1.8.1.

#### 3.1.6.1 The Relational Connectives

The master dictionary contains definitions of the usual relational connectives: =, #, <, >, =>, >=, <=, =<, EQ, NE, GT, GE, LT, and LE. All of these except # may be used in any combination (# must be used by itself). Note that all normal combinations are already defined. The form = < may be used as well as =<, for example. However, the form with the space between the connectives requires that two look-ups must be done, while the =< is retrieved in a single master dictionary reference. If you have a preference for the form <>, you may copy the item '#' in the master dictionary to the item '<>'. The operators are logically equivalent.

### 3.1.7 THE SPECIFIED VALUE AND ATTRIBUTE 7

It was noted that the contents of attribute 7 may affect the results of the selection processor. This is because attribute 7 is designed to produce output conversions. For this reason, the ACCESS compiler will execute an inverse conversion on the data values defined in the value string, so that the output conversion does not need to be done for each value in the file referenced by the data definition item. The compiler then throws away the contents of attribute 7.

The ACCESS compiler will not attempt to execute an F- or an A-correlative in attribute 7. These will be ignored.

There are conversions which will yield unexpected results, however, and should be avoided.

#### 3.1.7.1 Allowable Conversions - Date

Conversions which may be used are primarily the date and time conversions. The data conversion will take a date in display form and return it in internal form, which is a decimal number representing the number of days since December 31, 1967. In this case, you would not wish to execute a date output conversion in attribute 8, since it is unlikely that you would ever get a match. Note that an input date conversion will only transform the external form of a date into the internal form, and that an output conversion will only transform an internal date into the external form of the date. The only time that an input conversion is done in ACCESS is for the evaluation of values associated with selection phrases according to attribute 7.

It is preferable to do the date conversion associated with selection in attribute 7 because it only needs to be done once, at compile time, and because, if it is done in attribute 8 on each value, it will be necessary to remember the precise form which will result. Also, the form which derives from attribute 8 will be evaluated according to the alphanumeric form of an external date, rather than the numeric form of the internal date. Since the internal date is represented as an increasing integer, the less than and greater than relational connectives have the expected meaning of before and after. The external date does not have this characteristic. It is therefore advisable to store dates in files in the internal form.

#### 3.1.7.2 Allowable Conversions - Time

Time conversions are allowable. Time is represented in the machine as an integer which is the time since midnight in seconds. Again, it is preferable to use the attribute 7 form.

### 3.1.7.3 The Mask Conversions

In general, the forms MR, ML, and MD will treat only the scaling and decimal location characteristics available with these masks. Nothing else will be touched. This means that they will have an effect only if they are immediately followed by one or two numeric digits. Also, they will have an effect only on a value string which represents a number. If the value is a number, it is scaled and the decimal place is attended to. Remember that the internal form is an integer. That is, there is no decimal point in the number. If there are some numeric digits at the front of the value, then these will be taken as the number, and the rest of the value will be thrown away, unless the number is attended to in an attribute-8 F-correlative. If the first character of the specified value is not numeric, then the value string will be taken without modification.

Thus, if attribute 7 has a MR, ML, or MD conversion in it, numeric values are recommended. Note especially that masks of the form 'ML#20' and 'MR#10' have no effect.

### 3.1.7.4 Other Masking Functions

Since the functions of the form MCA, MCN and so on have the effect of stripping non-admissable characters from the string which they process, they should not be used.

The MCXD (convert from hex to decimal) and MCDX (convert from decimal to hex) conversions have inverse functions, so that they may be used in attribute 7 of a data definition item being used for selection. The MCXD will convert decimal to hex as an input conversion, and the MCXD will convert hex to decimal as an input conversion.

### 3.1.7.5 Translate Conversions

If there is a translate conversion in attribute 7, then it will be executed as an input conversion. This means that the first of the translate attribute mark count numbers in the translate syntax will be used. If the field is null, and if the translate found an item-id, it will return the item-id.

If the value specified does not yield an item-id, and if the translate option byte is an 'X', then the value for which the processor will search will be a null. If the option byte is a 'C', then the value for which the processor will search will be the specified value.

What the input translate will not do is search the file specified by the translate for an item which has the specified value in the correct attribute, and return the item-id as the value.

If there is a direct one-to-one correspondence between the source and destination items, then it is possible to have a set of translate elements within the file which are an inverse transformation. That is, if you supply the value generated by the output translation, and if that value is an item in the file which has as contents the item-id of the value which translates to the value supplied, then a translation in attribute 7 is valid. For example:

In a file called CUSTOMER,

```
     232                   Pacific Printing    (the item names)
001 Pacific Printing   001 232    (the translate references)
```

Then if attribute 7 of the data definition item CUSTTRANS is

```
TCUSTOMER;C;1;1
```

and the data file reference to Pacific Printing is '232', then

```
LIST FILENAME WITH CUSTTRANS = "Pacific Printing" ... CUSTRANS ...
```

will yield the desired result, because Pacific Printing will be translated into 232 for the selection, and 232 will be translated into Pacific Printing for output.

If the output translate function translates several different strings to a single output result, such that is would require an inverse function which is multi-valued, then a translate in attribute 7 is inappropriate, because only the first value found by the attribute 7 manipulation will be included in the resultant value string. In this case, the translate must be put in attribute 8, so that the processor is comparing the translated value to the value originally specified in the value string.

## 3.1.8 SUMMARY OF CONVERSIONS FOR SELECTION

It is generally a good idea to use the date and time conversion in attribute 7. The MCXD and MCDX conversions will work. The MR, ML, and MD conversions will only work in some cases.

The translate conversion will work if the data structure is just right.

The various other masks and conversions which have no natural inverse functions will tend to fail, and are not recommended.

The processor will not deal with A- and F-correlatives. In all cases, the contents of attribute 7 are discarded during the compilation process.

### 3.1.8.1 Forming Conditional Values

As noted previously, there are three special characters associated with value specification: '[', '^', and ']'. These are not optional and they are not modifiable. They have the following meanings:

| | |
|---|---|
| [ | stipulates that any leading string is acceptable. |
| ^ | stipulates that any character is acceptable in this position. |
| ] | stipulates that any trailing character is acceptable. |

The test point for these characters will always meet with success. For purposes of evaluation, the inclusion of a special character forces evaluation from left-to-right, on a character-by-character basis. For example:

= "[6" will accept any data value which terminates in a '6', such as

'6' or 'ABC6' or '123456'.

= "3^5" will accept any three-character string which begins with a '3' and ends with a '5', such as

'305' or '3A5' or '3#5.

= "6]" will accept any string which starts with a '6', such as

'6' or '6ABC' or '654321'.

= "[6]" will accept any string which contains a '6', such as

'6' or 'ABC6' or '6ABC' or 'ABC6XYZ'

= "[3^5]" will accept any string which contains any three-character string which starts with a '3' and ends with a '5', such as

'335' or '305XYZ' or 'ABC3X5' or 'ABC3X5XYZ'.

There are certain forms which will not work. If the '[' is used in the value specification, it must be the first character in the string, and it must be the only '[' in the string. If the ']' character is used in the string, it will terminate the specified string at that point. Any characters which may occur after a ']' will never be inspected. The ^ may be used anywhere, and any number of them may be included in the value specification. The form '^^^' may be used to retrieve all three-character strings, for instance, although there is a length conversion, 'L', which performs this function.

### 3.1.8.1.1 The Special Characters and the Relational Connectives

The special characters '[', '^', and ']' are used with the relational connectives < (less than), > (greater than) and =, since the other permutations can be derived from these.

The case of equality was shown in the preceding section. If the form:

```
WITH 2 < "5]"
```

is used, the test is on the first character, and is straightforward. If the form:

```
WITH 2 < "[5"
```

is used, the test is on the last character, and is straightforward. If the form:

```
WITH 2 < "[5]"
```

is used, then, if there is a '5' anywhere in the string, equality will be true, and inequality will fail. If there is no '5' in the data string, then the condition 'less than' will hold if the last character is less than the 5, and the condition 'greater than' will apply if the last character in the data string is greater than '5'.

Similarly, if the string of actual data specified is several digits long, the test which generates the type of equality will be as follows: if the process reaches the end of the data string when it is on the first real character of the test string, it will compare that character and yield a result as above. If the it is on a character other than the first real character in the specified string, it will generate the result expected if the compare were on the first k characters in the specified string against the last k characters in the data string. Other examples of inequality with [ and ]:

| | |
|---|---|
| "[567]" < "12486" | because 5 < 6. |
| "[567]" > "12484" | because 5 > 4. |
| "[567]" < "12457" | because 56 < 57. |
| "[567]" > "12455" | because 56 > 55. |
| "[567]" < "12568" | because 567 < 568. |
| "[567]" > "12566" | because 567 > 566. |

Equality will not occur unless all of the real character string is found in the data string.

### 3.1.8.1.2 Justification and Evaluation

If the attribute definition item is left-justified or if there is a special character in the value specification string, then comparison will proceed from left to right, and inequality will be declared as soon as characters in the same location in the two strings are different. The collating sequence is that of the ASCII character set, with the particular characteristic that numbers precede letters, and capital letters precede lowercase letters. An absolute null is less than any character, including an ASCII null. An absolute null occurs when the end of a string is reached, with the result that 'ABC' comes before 'ABC0'. Also note that blanks are the non-control character with the lowest value.

If the attribute definition item is right-justified, and there are no special characters in the string, and the data string and value string are numeric, then the test will be on the magnitude of the two numbers such that 12 > 2. If these were left-justified, 12 < 2 because 1 collates before 2. If the data are not numeric, then they will be compared in the usual left-to-right manner until either inequality is discovered, the strings terminate, or numeric fields are found. If both the data and the specified value are equal up to the start of numeric fields, then the numeric fields will be evaluated as binary integers and compared. If inequality is found, then the string with the smaller embedded integer is taken as less. If they are equal and both strings terminate at this point, then the strings are equal. If the strings continue with non-numeric data, the left-to-right process continues until inequality occurs or the strings terminate.

Note that for numeric data to sort in correct ascending order, and to print with flush right alignment, it must be right-justified. (An 'R' must be specified in line 9 of the attribute definition item.) Since alphabetic data will sort in alphabetical order whether it is left or right justified, an 'L' or an 'R' may be placed in line 9 of the attribute definition item. However, the usual form of printing for alphabetic data is flush left.

### 3.1.8.2 Value Strings Containing Many Value Phrases

It is possible to have selection-criteria based on more than one value. The relation associated with each value is the relational connective which immediately precedes the value. If there is no relational connective which precedes the value, then a default '=' will be inserted into the value string. The implicit relation between the value phrases is 'OR'. If the data value must pass both of two criteria, then there must be an 'AND' specified between the two value phrases. The examples which follow show ORed values with relational connectives. Note that the letter values are evaluated in ascending alphabetic sequence.

```
... WITH X "A""C""E""G"              is equivalent to

... WITH X = "A" OR = "C" OR = "E" OR = "G"

                                     which will succeed if

(data = "A") OR (data = "C") OR (data = "E") ...
```

where data in each case represents only one value which may be returned to the value comparison processor. Therefore,

```
IF  (data = "A") OR (data = "C") OR (data = "E") ...

THEN data is TRUE ELSE data is FALSE.
```

A data value is said to succeed if the test returns TRUE.

The cases of inequality are similar:

```
... WITH X < "A" > "C" # "E" <= "G"

                                     is equivalent to

IF  (data < "A") OR (data > "C") OR (data # "E")

    OR (data <= "G") ...

THEN data is TRUE ELSE data is FALSE.
```

(This particular case will succeed in all cases.)

### 3.1.8.2.1 The AND Connective within a Value String

If you desire to specify a range of values which will be acceptable, or a collection of conditions on a given data value such that they must all be true in order for the condition to succeed, then the specified values may be ANDed together. The required form is:

```
    ... "value" AND  {relational connective} "value"
```

For example:

```
    ... WITH X >= "A]" AND <= "C]"        will accept all values which start
                                          with A, B or C as in

    IF ((data >= "A]") AND (data <= "C]"))

    THE data is TRUE ELSE data is FALSE.

    ... WITH X = "1]" AND < "[5"          will have the effect of accepting all
                                          values which start with 1 and end
                                          with a character less than 5, as in

    IF ((data = "1]") AND (data < "[5"))

    THE data is TRUE ELSE data is FALSE.
```

### 3.1.8.3 The Evaluation of One Selection Item

An indefinite collection of value phrases may be ANDed together into one AND value specification phrase. Further, several AND value specification phrases may be ORed together into a value string.

Essentially, an AND phrase, which may consist of subconditions, acts as a single entity which can either pass or not pass. For an AND value specification phrase to pass, all elements must pass. For the ORed value specification phrases, if any element succeeds, then the selection criterion succeeds.

## 3.1.9 THE RELATIONSHIP OF SELECTION CRITERIA WITHIN A SENTENCE

As noted, if one ORed value specification allows a data element to pass, the selection criterion will pass. It is possible to have several selection criteria within a single sentence. The default connective between the selection criteria will be an OR, so that if any of the criteria pass, the item will pass. This may be modified by the selection modifiers NOT and EACH. The NOT modifier will cause the criterion to fail in the case that the value string succeeds and vice-versa. Therefore, the table of success and failure under the conditions of WITH, WITHOUT, WITH EACH, and WITHOUT EACH which was displayed for the case of existence also applies here. Note that the case of existence is equivalent to the value string # "", although using the explicit string is inefficient. In the following lists of failure and success conditions below, the form '# NULL' is replaced by the form 'IS TRUE', and the form '= NULL' is replaced by the form 'IS FALSE', for values returned from the value test processor.

Success conditions for WITH and its modifiers under test against a value-list:

```
    WITH attribute-name value-list             succeeds if

    (value1 is TRUE) OR (value2 is TRUE) OR (value3 is TRUE) OR ...

    WITHOUT attribute-name value-list          succeeds if

    (value1 is FALSE) AND (value2 is FALSE) AND (value3 is FALSE) ...

    WITH EACH attribue-name value-list         succeeds if

    (value1 is TRUE) AND (value2 is TRUE) AND (value3 is TRUE) ...

    WITHOUT EACH attribute-name value-list     succeeds if

    (value1 is FALSE) OR (value2 is FALSE) OR (value3 is FALSE) ...
```

Failure conditions for WITH and its modifiers under test against a value list:

WITH attribute-name value-list fails if

(value1 is FALSE) AND (value2 is FALSE) AND (value3 is FALSE) AND ...

WITHOUT attribute-name value-list fails if

(value1 is TRUE) OR (value2 is TRUE) OR (value3 is TRUE) ...

WITH EACH attribute-name value-list fails if

(value1 is FALSE) OR (value2 is FALSE) OR (value3 is FALSE) ...

WITHOUT EACH attribute-name value-list fails if

(value1 is TRUE) AND (value2 is TRUE) AND (value3 is TRUE) ...

### 3.1.9.1 Selection-Criteria AND Clauses

The term selection-criterion is of the form:

WITH {NOT} {EACH} attribute-name {value-list},

such that each criterion tests the data in one attribute against the optional value-list, and modifies it as specified, across the data values that are present within each item. An AND clause is of the form:

selection-criterion AND selection-criterion AND selection-criterion ...

The criterion for success of an AND clause is that each selection-criterion succeed (see the previous lists of success conditions).

### 3.1.9.2 Data Selection-Criteria

Data selection-criteria are made up of one or more selection-criterion that are ORed together. These may include any number of ORed selection-criterion that are not members of AND clauses. The condition for success of the data selection-criteria is that at least one of the selection criteria which are ORed together succeeds.

### 3.1.9.3 Item Selection-Criteria

The condition for item selection is that the item-id tests succeed, and that the data selection-criteria succeed. In other words, the item-id test is implicitly ANDed with the data selection test.

# ASCII codes A

The ASCII codes used by the PICK System are:

| DEC | Hex | Character | DEC | Hex | Character |
|---|---|---|---|---|---|
| 0 | 0 | NULL | 36 | 24 | $ |
| 1 | 1 | SOH | 37 | 25 | % |
| 2 | 2 | STX | 38 | 26 | & |
| 3 | 3 | ETX | 39 | 27 | ' |
| 4 | 4 | EOT | 40 | 28 | ( |
| 5 | 5 | ENQ | 41 | 29 | ) |
| 6 | 6 | ACK | 42 | 2A | * |
| 7 | 7 | BEL | 43 | 2B | + |
| 8 | 8 | BS[1] | 44 | 2C | , |
| 9 | 9 | HT[1] | 45 | 2D | - |
| 10 | A | LF[1] | 46 | 2E | . |
| 11 | B | VT[1] | 47 | 2F | / |
| 12 | C | FF[1] | 48 | 30 | 0 |
| 13 | D | CR[1] | 49 | 31 | 1 |
| 14 | E | SO | 50 | 32 | 2 |
| 15 | F | SI | 51 | 33 | 3 |
| 16 | 10 | DLE | 52 | 34 | 4 |
| 17 | 11 | DC1 | 53 | 35 | 5 |
| 18 | 12 | DC2 | 54 | 36 | 6 |
| 19 | 13 | DC3 | 55 | 37 | 7 |
| 20 | 14 | DC4 | 56 | 38 | 8 |
| 21 | 15 | NAK | 57 | 39 | 9 |
| 22 | 16 | SYN | 58 | 3A | : |
| 23 | 17 | ETB | 59 | 3B | ; |
| 24 | 18 | CAN | 60 | 3C | < |
| 25 | 19 | EM | 61 | 3D | = |
| 26 | 1A | SUB | 62 | 3E | > |
| 27 | 1B | ESC | 63 | 3F | ? |
| 28 | 1C | FS | 64 | 40 | @ |
| 29 | 1D | GS | 65 | 41 | A |
| 30 | 1E | RS[1] | 66 | 42 | B |
| 31 | 1F | US[1] | 67 | 43 | C |
| 32 | 20 | SPACE | 68 | 44 | D |
| 33 | 21 | ! | 69 | 45 | E |
| 34 | 22 | " | 70 | 46 | F |
| 35 | 23 | # | 71 | 47 | G |

| DEC | Hex | Character | DEC | Hex | Character |
|---|---|---|---|---|---|
| 72 | 48 | H | 104 | 68 | h |
| 73 | 49 | I | 105 | 69 | i |
| 74 | 4A | J | 106 | 6A | j |
| 75 | 4B | K | 107 | 6B | k |
| 76 | 4C | L | 108 | 6C | l |
| 77 | 4D | M | 109 | 6D | m |
| 78 | 4E | N | 110 | 6E | n |
| 79 | 4F | O | 111 | 6F | o |
| 80 | 50 | P | 112 | 70 | p |
| 81 | 51 | Q | 113 | 71 | q |
| 82 | 52 | R | 114 | 72 | r |
| 83 | 53 | S | 115 | 73 | s |
| 84 | 54 | T | 116 | 74 | t |
| 85 | 55 | U | 117 | 75 | u |
| 86 | 56 | V | 118 | 76 | v |
| 87 | 57 | W | 119 | 77 | w |
| 88 | 58 | X | 120 | 78 | x |
| 89 | 59 | Y | 121 | 79 | y |
| 90 | 5A | Z | 122 | 7A | z |
| 91 | 5B | [ | 123 | 7B | { |
| 92 | 5C | \ | 124 | 7C | : |
| 93 | 5D | ] | 125 | 7D | } |
| 94 | 5E | ^ | 126 | 7E | ~ |
| 95 | 5F | _ | 127 | 7F | DEL |
| 96 | 60 | ` | . | | |
| 97 | 61 | a | . | | |
| 98 | 62 | b | . | | |
| 99 | 63 | c | 251 | FB | SB[2] |
| 100 | 64 | d | 252 | FC | SVM[2] |
| 101 | 65 | e | 253 | FD | VM[2] |
| 102 | 66 | f | 254 | FE | AM[2] |
| 103 | 67 | g | 255 | FF | SM[2] |

[1]For special use on LSI-11 and -12 terminals:

| | | | |
|---|---|---|---|
| BS | Cursor Backspace | FF | Cursor Forward |
| HT | Cursor Tab | CR | Cursor Carriage Return |
| LF | Cursor Down | RS | Cursor Home |
| VT | Cursor UP | US | Cursor New Line |

[2]For special use by PICK:

| | |
|---|---|
| SB | Start buffer |
| SVM | Secondary value mark (displays \) |
| VM | Value mark (displays ]) |
| AM | Attribute mark (displays ^) |
| SM | Segment mark (displays _) |

INDEX