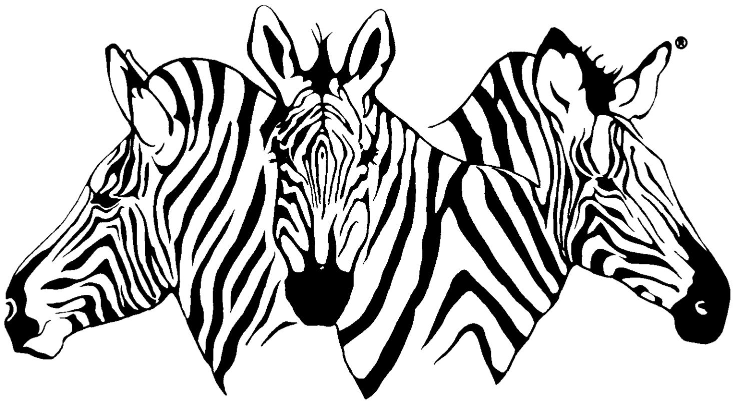
introduction to PICK TCL and file structure

88A00782A01



ZEBRA® BY GA

RECORD OF REVISIONS

Title: Introduction to PICK TCL and File Structure

Document No. 88A00782A01

Date	Issue
March 1984	Original Issue

NOTICE

introduction to PICK TCL and file structure

88A00782A01

RECORD OF REVISIONS

Title: Introduction to PICK TCL and File Structure

Document No. 88A00782A01

Date	Issue
March 1984	Original Issue

FOREWORD

The PICK Terminal Control Language (TCL) is the primary link between the user, his files and what he wishes to do with the files. This document focuses on TCL and the file structure.

It is recommended that "new" users of the ZEBRA/PICK system become familiar with these sbjects before extensive use of the processor family is undertaken.

The following documents cover processors that can be called for carrying out different operations with file data:

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
88A00776A	PICK ACCESS Reference Manual
88A00777A	PICK SPOOLER Reference Manual
88A00778A	PICK BASIC Reference Manual
88A00779A	PICK EDITOR Referance Manual
88A00780A	PICK PROC Reference Manual
88A00781A	PICK RUNOFF Reference Manual
88A00783A	PICK JET Word Processor Guide

TABLE OF CONTENTS

introduction 1

PICK is a user-friendly data base management system that allows multiple users to instantly create, retrieve and update data stored in on-line files. Users can communicate with the system locally, or remotely, through keyboard terminals.

One of the outstanding features of PICK is the ability to customize the vocabulary for each user. Since the vocabulary words (called ´verbs´) reside in the individual user´s Master Dictionary (MD), the vocabulary can be changed as desired without affecting the other users. Communication statements can, therefore, be in the user´s language; and, those statements can be tailored with words and phrases that are unique to the user´s particular business field.

1.1 KEY WORDS

Before reading descriptions of the PICK verbs and their processors, the new user should be introduced to key words that have special meanings. The following words are not in strict alphabetic sequence. Words that are closely related are grouped together to be more meaningful.

DICTIONARY
A dictionary is a "special" file. A dictionary contains items which define the attributes which make up the (other) items in the data file associated with it. A dictionary may also contains an item(s) which gives the size and (disk) location of its associated data file(s). Every data file must have its dictionary, although in some cases, it may share its dictionary with similar files.

FILE
Physically, a file consists of a chain of disk areas in which its items are stored. Although a dictionary is a file, usually "file" refers to a data type file and dictionaries are called "dictionaries". Usually a file contains similar types of items such as inventory or customer files, but some files, such as program or word processing files, do contain dissimilar items.

ITEM
A file record made up of attribute values. Items, in turn, make up a file. Items in an inventory file could contain the part number, cost, vendor, quantity on hand, date of last purchase, etc.

ITEM-ID

The name of an item in a file or the attribute position zero (0) in a dictionary. Part number is usually the item-id in an inventory file; customer number could be the item-id in an accounts receivable file. An item-id may consist of numbers and/or letters or other characters except certain characters reserved as "system delimiters". Blanks should be avoided, if possible, but are allowed in an item-id.

ATTRIBUTE

A specific piece of data which, along with other attributes, makes up an "item". Name, address, city, state, zip could be 5 attributes of the items in a name and address file.

Attribute numbers and and attribute names are not stored in an item, only the values. The attribute number gives the relative position of the value in the stored item. This attribute position number is also called the AMC of the attribute, meaning Attribute Mark Count.

An attribute is also referred to as a field.

FRAME

The disk storage on a PICK system is divided into 512 byte (character) sections called "frames". Each frame is numbered and can be accessed directly by the system using this number which is called the frame-id or FID.

GROUP

One or more disk frames linked together. The number of groups which are reserved for a file is specified by the "modulo".

MODULO

The modulo is the number of "groups" of disk frames reserved for a file. The modulo is specified at the time a file is created and is based upon an estimate of the number of characters which will be contained in the file.

SEPARATION

This is the number of frames in a group. Separation times the modulo gives the number of frames reserved for a file. Separation is usually "1", although under special circumstances, it may be more. Separation is specified at the time the file is created, but the system will automatically add frames to the group when and if needed.

terminal control language 2

2.1 INTRODUCTION TO TERMINAL CONTROL LANGUAGE

Terminal Control Language (TCL) is the primary interface between the user and the system. It is from TCL that all other processors (EDITOR, ACCESS, BASIC, PROC, etc.) are invoked. The TCL processor is automatically entered at LOGON and whenever a particular process run from TCL (such as an ACCESS or BASIC process) is complete. The TCL prompt is a ´>´.

A TCL statement calls into effect one of the TCL verbs (action-initiating commands) residing in the user´s Master Dictionary (MD) which either performs specified functions or invokes other processors to perform specified functions. For example, the TIME verb prints the current time and date on the terminal, while the RUN verb invokes the BASIC run-time processor which "runs´ the specified BASIC program. The general form of a TCL statement is:

```
TCL prompt character     Options enclosed in         Carriage return,
!                      Parentheses (must be at end)    (or line-feed)
!                                 !                         !
>Verb { . . . Parameters . . . } { (Options) } {[cs_]} [CR]
           !                                         !
           !                                         !
Parameters as required by the       Optional continuation character
specific verb.                      (Control-Underline or control-_).
```

The user is at the TCL level in the system when the system "prompts" with a ">" character. The ">" is printed at the far left on the terminal and indicates that the system is awaiting input from the terminal.

The TCL verbs belong to three major categories. Type I verbs perform specified functions, but do not access data in files; the TIME verb mentioned above is a Type I verb. Type II verb functions involve the accessing of data in files; the RUN verb mentioned above is a Type II verb. The third category is made up of ACCESS verbs, which are discussed in the ACCESS Manual.

Note that the control character for the optional continuation character will vary on different terminals, however, whatever control character corresponds to ASCII US (hex 1F) will perform the function on all terminals.

The user may create any number of synonyms for the verb definition items (and may even remove the predefined verb definition items), thereby creating his own vocabulary. Synonyms may be created by copying the verb definition item into another item with the desired name as the item-id.

A TCL statement consists of the TCL verb, any other parameters (words, file-names, options, etc.) that the specific verb may require, followed by a carriage return or line feed (shown as [CR] in the documentation). No action is initiated until the [CR] is input.

All TCL statements may have an "options" entry as the last parameter; options are single alphabetic characters, and/or a single or double number of the form "n" or "n-m", where n and m may be decimal, or hexadecimal if preceded by a period (.). The entire option string is enclosed in parentheses. Options affect the operation of each verb in a unique way. General options are "P" for routing data to the line printer and "N" for inhibiting the end-of-page wait at the terminal. Multiple options may be separated by commas for clarity. Examples of TCL statements are:

```
>TIME [CR]

>DUMP L,5000-5003 [CR]

>RUN BP LISTU (P) [CR]

>SORT INVENTORY WITH QUANTITY.ON.HAND > "200" BY QUANTITY.ON.HAND [cs_] [CR]
: DESC PRICE QUAN EXT.PRICE (PI) [CR]   (note options on second line!)
```

During the entry of the TCL statement, certain editing functions are available to the user. A control-H ([cH]) is used to BACKSPACE over the last character input. Normally, the terminal will also physically backspace the cursor or carriage to indicate that the last entered character has been deleted. A control-X ([cX]) may be entered to DELETE entirely the last entered line; a new line is initiated at the terminal by the system. A control-W ([cW]) may be used to backspace over the last WORD. A control-R ([cR]) may be used to RETYPE the last line. Following is a summary of TCL editing functions:

Character	Editing Function	Comments
Carriage Return [CR] (or line feed) [LF]	End of line.	System will take action on TCL statement.
Control-H [BS]	Backspace over last character.	No action if at left margin. Character echoed by system may be set by the TERM command.
Control-W [ETB]	Backspace over last word.	No action if at left margin.
Control-X [CAN]	Delete last line.	No action if at left margin; new line will be started otherwise.
Control-R [DC2]	Retype last line.	
Control-Underline [US]	Line continuation character.	Must be followed immediately by a [CR]. TCL will prompt with a ":" for next line of input. May be used repeatedly.

NOTE: The first five functions listed are system-wide editing functions and are applicable whenever the system requests data input from the user's terminal. The last function listed is usable only by TCL and specific other processors that allow multiple-line input. The control characters used to generate a particular function will vary from terminal to terminal. However, whatever control characters correspond to the ASCII characters given in square brackets will perform the corresponding function on all terminals.

2.2 TCL TYPE I INPUT STATEMENTS

TCL Type I verbs do not access a file. The format of the TCL statement is unique to the specific verb (i.e., there is no general form of the TCL statement using this type of verb).

A TCL-I input statement must begin with a TCL-I verb and end with a carriage return, and may sometines include various parameter specifications. Some TCL-I verbs are listed in Table 2-1. For further information regarding these verbs, refer to Utilities and Spooler Manuals.

Table 2-1. Some TCL-I Verbs

Verb	Description
BLOCK-PRINT	Sends block characters to spooler.
CHARGES	Prints current computer usage.
CHARGE-TO	Keeps track of computer usage.
MESSAGE	Communicates to other users.
MSG	Same as MESSAGE.
OFF	Terminates user´s session.
P	Inhibits printing at terminal.
SLEEP	Puts a terminal to "sleep" for a specified time, or until a specified time.
SP-ASSIGN	Sets up assignment status for the spooler.
SP-STATUS	Gives spooler and line printer status.
T-ATT	Attaches magnetic tape unit.
T-DET	Detaches the magnetic tape unit.
TABS	Sets tabs for input or output.
TERM	Displays terminal characteristics.
TIME	Prints current time and date.
WHAT	Prints current system parameters.
WHO	Prints the line number and account name to which any terminal is logged on.

2.3 TCL-II INPUT STATEMENTS

TCL Type II verbs allow access to a specified file. The format for forming a TCL-II input statement is more restrictive than for an ACCESS statement (refer to the ACCESS Manual). The advantage gained by this restricted format is an enhancement in processing speed since statement parsing is quicker.

A file-name (or DICT file-name) must immediately follow the TCL-II verb. Item selection is more restricted than in ACCESS statements, since each item-id must be explicitly named in the statement (or, alternatively, all items may be specified via use of the asterisk (*) character).

The general form of a TCL-II statement is:

```
    TCL-II verb   file being accessed      options enclosed in parentheses
      !               !                          !
    >Verb {DICT } file-name {item-list} { (options) } {[cs_]} [CR]
           !                   !
     Specifies access to      specific items, or "*"; if
     dictionary of file       omitted, items are obtained
                              from a previously specified
                              select-list
```

The file-name specifies the desired file. The DICT option specifies the dictionary portion of the file. The item-list is made up of one or more item-ids, separated by one or more blanks. If an item-id contains embedded blanks or parentheses, it must be surrounded by single quotes. All items in a file may be specified by using an asterisk (*) character as the item-list. Options, if specified, must be enclosed in parentheses at the end of the input line. Multiple options may be separated by commas. The specified options are passed to the appropriate TCL-II processor. Samples of TCL-II input statement:

```
    >COPY MD LIST SORT (T) [CR]

    >ED INVENTORY 123-987 [CR]

    >RUN BASIC/PROGRAMS UPDATE/LISTS (D,P) [CR]

    >COMPILE BASIC/PROGRAMS ALPHA (LMN) [CR]
```

The item-list may be omitted entirely, if the TCL-II statement is preceded by a statement that generates a "select-list"; the item-ids are then obtained from this preselected list. Statements that generate select-lists are SELECT, SSELECT, QSELECT, and GET-LIST, and are described in the ACCESS Manual.

Some TCL-II verbs are:

Verb	Description
COMPILE	Compiles a BASIC program.
CATALOG	Catalogs a BASIC program.
COPY	Copies data files and dictionaries.
ED or EDIT	Evokes the EDITOR processor.
RUN	Executes a BASIC program.
RUNOFF	Evokes the text processor.

For further information regarding these verbs, refer to the BASIC, EDITOR, RUNOFF, and UTILITIES Manuals.

2.4 LOGGING ON AND OFF THE SYSTEM

The Logon processor provides a facility for initiating a user's session by identifying valid users and their associated passwords. The Logoff processor is used to terminate the session and should always be evoked via the verb OFF when the user wishes to terminate. These processors can accumulate accounting statistics for billing purposes and also will associate the user with his privileges and security codes.

2.4.1 LOGGING ON TO THE SYSTEM

The user may log on to the PICK Operating System when the following message is displayed:

```
LOGON TO THE GA XX ZEBRA AT 00:00:00
PLEASE ENTER ACCOUNT NAME >
```

NOTE

The actual form of this message will vary from system to system, since the message format is obtained from an entry called "LOGON" in the SYSTEM Dictionary.

The user then enters the account name followed by a carriage return. The system validates the user's identification (account name) against the entries in the SYSTEM Dictionary; if it is illegal, the following message is returned:

```
USER-ID?
```

The user must then re-enter the account name.

If a password has been established, there will be a prompt:

```
PASSWORD:
```

He then enters the password followed by a carriage return. (The password is not echoed back to the terminal when it is entered.) If a valid password is not entered, the system will display:

```
PASSWORD?
```

The user must then reenter the account name and password.

The user may also enter the password at the time the account name is entered by entering the account name, followed by a comma, followed by the password, and then a carriage return. A password entered in this fashion will be displayed on the terminal.

If the user has successfully logged on to the system (i.e., both the account name and the password have been accepted), the following message is displayed:

```
<<< R80 GENERAL AUTOMATION   REV: m.n >>>
<<< 00:00:00   ZEBRA              date>>>
.
.
>                            <.. TCL prompt.
```

where:

R80	is the PICK Operating System release level.
>	is the TCL prompt character, which indicates that the user may now enter any valid TCL level command.

2.4.2 LOGGING OFF OF THE SYSTEM

Logoff is achieved by entering the word OFF, either at the TCL level or at the DEBUG level. A message indicating the connect time (i.e., number of minutes that the user was logged on) and the appropriate CPU units will be displayed. The system then displays the LOGON PLEASE message and waits for the next user session to be initiated. The general form of the logoff message is as follows:

```
< CONNECT TIME = n MINS.; CPU = m UNITS; LPTR PAGES = x >
<       LOGGED OFF AT time      ON  date                >
```

where:

n	is the number of minutes of connect time
m	is the number of CPU units
x	is the number of line printer pages generated
time	is the current time
date	is the current date

The CPU units represent usage of the CPU; it is in tenths of a CPU second.

2.5 ADDITIONAL LOGON FUNCTIONS: LOGTO, CHARGE-TO, AND CHARGES

The LOGTO verb allows the user to log to another account faster than by going through the OFF and LOGON process. The CHARGE-TO verb allows the user to charge a particular logon session to a specific charge number or name; the CHARGES verb displays the charge statistics for the current logon session. An example of the usage of these verbs:

```
    *    PLEASE ENTER ACCOUNT NAME > SMITH,XYZ [CR]


      <<< R80 GENERAL AUTOMATION   REV: m.n >>>
      <<< time       ZEBRA            date    >>>

    *   >WHO [CR]
        7 SMITH

    *   >TIME [CR]
        09:17:00   11 DEC 1982

    *   >LOGTO JONEA [CR]

        USER-ID?

    *   >LOGTO JONES [CR]
    *   PASSWORD:  ABC [CR]       NOTE:  The password is not displayed.)

     < CONNECT TIME = 3 MINS.; CPU = 11 UNITS, LPTR PAGES= 0 >

    *   >WHO [CR]
         7 JONES

    *   >CHARGE=TO A001 [CR]

     < CONNECT TIME = 0 MINS.; CPU = 7 UNITS, LPTR PAGES= 0 >

    *   >WHO [CR]
         7 JONES*A001

    *   >CHARGES [CR]

     < CONNECT TIME = 0 MINS.; CPU = 8 UNITS, LPTR PAGES= 0 >

    *   >CHARGE-TO [CR]

     < CONNECT TIME = 0 MINS.; CPU = 9 UNITS, LPTR PAGES= 0 >

    *   >WHO [CR]
         7 JONES
```

The general form of the LOGTO verb is as follows:

```
    LOGTO account-name {,password}
```

where:

account-name	is that of the other new account that the user wishes to logon to.
password	is the valid password for the other account.

If the account has a password defined and the user has not entered a password, the message:

```
    PASSWORD:
```

will be displayed and the password may then be entered. This password will not be displayed on the terminal.

If the account-name is illegal, the message:

```
    USER-ID?
```

will be printed and the user will be back at TCL.

If the password is incorrect, the message:

```
    PASSWORD?
```

will be displayed, and the user will be back at TCL.

If the account-name and password are both correct, the current logon session will be terminated by updating the accounting file with the appropriate statistics, and a new session started. The message:

```
    < CONNECT TIME = n MINS,;  CPU = m UNITS, LPTR PAGES= x >
```

will be displayed.

Note that it is possible to enter the account name and password separated by a comma on one line, however, this entry method will display the password on the terminal.

Also, the tape unit and line printer will be detached, if the user had them attached to his line prior to the LOGTO.

The CHARGE-TO statement has the following general form:

```
    CHARGE-TO {text}
```

where:

    text   is any sequence of non-blank characters.

This statement will cause the current logon session to be terminated and the account file to be updated with the appropriate statistics; a new session is started, with the new user identification of the form:

```
    account-name*text
```

where:

    text   is as specified in the CHARGE-TO statement.

This allows the user to charge his logon sessions to specific names or numbers. If "text" is null in the CHARGE-TO statement, the user identification will revert to the form "account-name" alone.

This CHARGE-TO statement will cause the following message to be displayed:

```
   < CONNECT TIME = n MINS.;  CPU = m UNITS;  LPTR PAGES= x >
```

The CHARGES verb has the following form:

```
    CHARGES
```

This will display the logon statistics with the following message:

```
   < CONNECT TIME = n MINS.;  CPU = m UNITS;  LPTR PAGES= x >
```

2.6 THE LOGON PROC AND GENERAL SYSTEM MESSAGES

Upon logon, PICK allows the execution of a PROC in the user's MD with an item-id identical to the user's identification. PICK also allows a general message to be sent to each user when he logs on to the system. Material in this section will be covered in Section 2.9 of this manual and in the PROC Manual.

When the user has logged on to his account, PICK permits the automatic execution of a PROC in the user's MD whose item-id is the same as the user's identification. That is, the Master Dictionary of the account will be searched for a PROC matching the identification which was used to log on to the account; if it is found, it will be executed.

Typically, the Logon PROC is used to perform standard functions that are always associated with the particular user's needs. For example, setting of terminal characteristics could be performed by the Logon PROC. When the user logs on to the system, his terminal characteristics are set to the initial conditions as follows:

	Terminal	Printer
Page Width:	79	132 characters
Page Depth:	24	60 lines
Line Skip:	0	
LF Delay:	1	
FF Delay:	2	
Backspace:	8	
Term Type:	M	

These conditions can subsequently be displayed and altered by the TCL verb TERM. As an example, assume that the PROC listed as follows:

item 'SMITH' in MD of user SMITH

```
001  PQ
002  HTERM 118,44,7,6
003  P
004  X***  TERMINAL CHARACTERISTICS SET  ***
```

is stored as item SMITH in the user's MD. If the user's identification is the word SMITH, then the SMITH PROC will be executed automatically every time the user logs on, and the user's particular terminal characteristics will automatically be set. This is illustrated on the following page.

```
PLEASE ENTER ACCOUNT NAME > SMITH,XYZ  [CR] <------------- Logon sequence.

**** NOTE: SYSTEM PRINTER WILL BE DOWN AT 12:00   <- General system
**** TODAY FOR ROUTINE MAINTENANCE UNTIL 2:00PM      message.

<<< R80 GENERAL AUTOMATION   REV: m.n >>>
<<< time       ZEBRA              date   >>>

***   TERMINAL CHARACTERISTICS SET   *** <------- Message from SMITH
                                                  PROC.
> <---------------------------------------------- TCL prompt
                                                  character.
```

A general system message may be stored in the special item "LOGON" of the system error-message file, ERRMSG. This file and the formats of the data in it are described in the PICK Operator Guide. The general system message will be printed immediately before the standard logon message

"<<< R80 GENERAL AUTOMATION   REV: m.n >>>".

Typically, this facility is used to broadcast system messages relating to everybody who logs on. Note that the item "LOGON" must exist in the ERRMSG file even if there is to be no system-wide message; in this case, it will be a null item.

2.7 SETTING TERMINAL/LINE PRINTER CHARACTERISTICS: TERM AND SET-TERM

Terminal and/or line printer characteristics may be displayed or set by a process via the TERM command. The general form of the TERM command is as:

```
TERM {a,b,c,d,e,f,g,h,t}
```

where:

a terminal line length (characters per line. Must be in range 16<a<140).

b number of print lines per page on the terminal.

c number of blank lines per page on the terminal (b + c = page length).

d number of delay or idle characters following each carriage return or line feed. Used for terminals that require a pause after a carriage return or line feed (the CPU generates characters faster than the terminal can accept them).

e number of delay characters following each top-of-form.

If e = 0	No form-feed character sent to printer or terminal.
If e ≠ 0	Form-feed character also output before each page.
If e = 1	Form-feed character sent to printer, not to terminal.
If e > 1	Form-feed character also sent to terminal at beginning of each page, and that many delay or idle characters also sent to give terminal time to settle after the form-feed. The character sent to the printer is always X'0C' (ASCII FF).

f backspace character. An ASCII backspace (control-H) is always input to backspace over (or erase) the last character that was input. User may set actual character echoed to terminal to accommodate terminals that cannot physically backspace, or have a backspace character other than the ASCII backspace.

g line printer line length.

h line printer page length.

t terminal type code; changes the form-feed character sent by the system to match terminal requirements and also sets the appropriate cursor addressing for BASIC cursor functions. The letters used for supported terminals are:

L	LEAR-SIEGLER ADM-11, ADM-12	T	TELEVIDEO 925, 950
		V	ADDS VIEWPOINT
M	AMPEX, DIALOGUE 80	X	No cursor addressing functions needed

Individual parameters may be null (specified by two adjacent commas in the TERM command). If so, the previously defined parameter remains in force. A TERM command without a parameter list causes display of the current characteristics. To function properly, the t parameter must be the last element in any TERM string. It may be the only element if no other elements are to be changed. The other parameters are positional, however.

The general form of the SET-TERM is the same as the TERM command

```
SET-TERM {a,b,c,d,e,f,g,h,t}
```

The SET-TERM verb sets the default printer and terminal characteristics for subsequent logons on all terminals. This verb is present only on accounts with SYS2 privileges. A sample usage of TERM:

```
>TERM [CR]

             TERMINAL PRINTER
PAGE WIDTH:     79      132
PAGE DEPTH:     24       60
LINE SKIP :      0
LF DELAY  :      1
FF DELAY  :      2
BACKSPACE :      8
TERM TYPE :      M
```

Standard terminal characteristics set for the DIALOGUE 80 terminal.

```
>TERM ,,,,,,120,48 [CR]
>TERM [CR]

             TERMINAL PRINTER
PAGE WIDTH:     79      120
PAGE DEPTH:     24       48
LINE SKIP :      0
LF DELAY  :      1
FF DELAY  :      2
BACKSPACE :      8
TERM-TYPE :      M
```

Resets the line printer page size to 120x48.

2.8 SETTING TAB STOPS: TABS

Tab stops may be set with the TABS statement. The general form of the TABS command is as follows:

```
    TABS I or O  n1,n2,n3......
```

or

```
    TABS I or O  {S}
```

where:

I or O	may be set for input or output, by setting either "I" or "O" following the TABS verb.
n1, n2, etc.	are up to 15 tab-stop positions; they must be in ascending numerical sequence.
S	recalls previously set tabs.

Tabs set for input are then available at any time that the system requests input from the terminal. By entering a control-I ([cI]), the system will space over to the next tab-stop position, if any. If there are no more tab-stop positions, the [cI] is ignored (control-I is also generated by the TAB key on some terminals). The tab stops set by the TABS I statement are identical to those set by the TB statement in the EDITOR.

Tabs set for output are only useful for terminals that have a physical tabbing capability. Do not set output tabs for a CRT that does not have this feature! If output tab stops are set, the system will replace blank sequences in any output generated by the system by an appropriate tab character [cI], thus reducing the data output. The user must also set up the physical tab stops on the terminal to correspond to those set in the TABS O statement. On many terminals, this entails positioning the carriage and entering a set-tabs sequence from the keyboard.

Input or output tab stops may be disabled by entering "TABS I" or "TABS O" respectively. Previously set tab stops may then be recalled by entering "TABS I S" or "TABS O S" for input and output tab stops respectively.

Currently set tab stops can be displayed by entering "TABS" alone.

Examples of TABS:

```
>TABS I 4,8,12,16,20,24,28 [CR]     (sets input tab stops)

>TABS 0,10,20,30,40,50,60 [CR]       (sets output tab stops)

>TABS [CR]                           (displays current tab stops)


         1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890
   I   I   I   I   I   I   I
         0         0         0         0         0         0

>TABS 0 [CR]                         (turns off output tab stops)

>TABS [CR]

         1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890
   I   I   I   I   I   I   I

>TABS I 5,15,20,40,50 [CR]

>TABS 0 S [CR]                       (recalls previous output tab stops)

>TABS [CR]

         1         2         3         4         5         6         7
1234567890123456789012345678901234567890123456789012345678901234567890
    I         I    I                   I         I
         0         0         0         0         0         0
```

2.9 MISCELLANEOUS UTILITY VERBS: TIME, SLEEP, WHO, MSG AND ECHO

The TIME statement displays the current system time and date; its general form is:

```
    TIME
```

For example,

```
    >TIME [CR]
    09:11:23  11 DEC 1982
```

The WHO statement is used to display the account-name that a terminal is logged on to; its general form is:

```
    WHO {n}
```

If WHO is entered without the "n", the line number (channel number) of the user's terminal is displayed, along with the account-name that he is logged on to. If the "n" is specified, the same data is displayed for line number "n", where n ranges from 0 to the maximum number of lines on the current system. If the line is non-existent, or if no user is logged on to that line, the account-name is replaced with "UNKNOWN".

Lines which have the name UNKNOWN and which are actively processing suggest that something strange is happening on the system, unless they are spoolers or you have an account on the system by the name of UNKNOWN.

Not only may one specify a line, but one may specify a range of lines as well. Any non-numeric character will cause WHO to display all lines and their logon name.

For example,

```
    >WHO [CR]
    07 SMITH                (this is line-number 7, logged on to "SMITH")

    >WHO 0 [CR]
    00 SYSPROG              (line number 0 is logged on to SYSPROG)

    >WHO 11 [CR]
    11 UNKNOWN              (line number 11 is not logged on)

    >WHO * [CR]              (displays accounts using all lines; lines
                             which are not logged on display UNKNOWN.)

    >WHO 1-3 [CR]

          01 JOHN
          02 SYSPROG
          03 UNKNOWN

    >WHO 'SYSPROG' [CR]      (displays all lines logged onto the SYSPROG
                             account)
```

The SLEEP verb is used to put a terminal to "sleep", that is, to enter a quiescent state for a specified period of time, or until a specified time. Its general form is:

```
    SLEEP x
```

where:

x is either a decimal number specifying the number of seconds to sleep, or is of the form: "hh:mm:ss" or "hh:mm:", specifying a time in 24-hour format until which to sleep.

SLEEP is useful to cause a terminal to wait until some time to run a task; for instance, the FILE-SAVE may be run at 23:00 (11:00PM) every night. For example:

```
    >SLEEP 100 [CR]        (terminal will sleep for 100 seconds)

    >SLEEP 23:00 [CR]      (terminal will wake up at 11:00 pm)
```

The form of SLEEP with a wake-up time is usable for a maximum of 24 hours.

The MSG (or MESSAGE) statement allows one user to send a message to another. The general form of the MSG statement is:

```
    MSG account-name ...... Message text ......     [CR]
```

where:

account-name is the name that the other user is logged on to and the text of the message follows

The message text is not edited in any way; there is no "options" parameter in the MSG statement.

Note that ALL users who are logged on to the specified account-name will receive the message.

Users with system level 2 privileges can broadcast a message to all users by substituting an asterisk (*) for the "account-name" in the MSG statement. This message will be received by the user´s terminal also.

A user who was entering data when a message is received may lose up to 16 characters due to the interference of the message; he should use the retype line character ([cR]) to see exactly what data is left.

Examples:

```
    >MSG JONES*A0001 WHAT'S THE STATUS OF THE INVENTORY REPORT??? [CR]

    >MESSAGE JONES HELLO THERE!"%%%%'''% [CR]

    USER NOT LOGGED ON                   (JONES is not logged on).

    >MSG * SYSTEM FILE-SAVE WILL START IN 5 MINUTES!!! [CR]
```

The MESSAGE and MSG verbs may be used to direct a message to a particular line as well as to a particular user by preceding the line number with an exclamation mark (!). This form of the verb will send messages to terminals which are not logged on. Further, the user may send a message to all lines, signed on or not, through a special form of this verb. The following outlines the syntax:

MESSAGE !12 HELLO	Send the message 'HELLO' to the user on line 12.
MSG !* SIGN OFF NOW	Sends a message to all terminals connected to the computer.

The ECHO verb suppresses printing on the terminal The general form is:

```
    ECHO {(I)} or {(L)}
```

The function of this verb is to toggle the switch in each user's PIB indicating whether or not characters typed in are to be echoed to the terminal. Thus, typing ECHO in normal mode will cause all further typing to be echo-suppressed. Similarly, typing ECHO in suppressed mode will cause echoing to resume. The user may also force a particular echo status: ECHO (I) will force echo suppression, ECHO (L) will force echoing.

2.10 MISCELLANEOUS UTILITY PROCS

This section describes commonly used utility PROCs. For a complete description of all utility verbs and PROCs, see the Utilities Manual.

CT PROC

The CT PROC is invoked by typing:

```
    CT file-name item-list {options}
```

The item(s) specified will be copied to the terminal. Options recognized by the COPY verb may be added.

LISTACC PROC

The LISTACC PROC is invoked by typing:

```
    LISTACC {account-name(s)}
```

This PROC lists accounting data for the account-name(s) specified. If no account-name(s) are specified, accounting data for all users will be listed.

LISTCONN PROC

The LISTCONN PROC is invoked by typing:

```
    LISTCONN {file-name} {(P)}
```

This PROC sorts all connectives in any dictionary and lists them on the terminal (or the line printer if P is specified). If no file-name is specified, the account's master dictionary will be used.

LISTDICT PROC

The LISTDICT PROC sorts all attribute synonym definition items in any dictionary and lists them on the terminal (or the line printer if P is specified). If no file-name is specified, the account's master dictionary will be used. The general form is:

```
    LISTDICTS {file-name} {(P)}
```

LISTFILES PROC

The LISTFILES PROC sorts all file or file synonym definition items in any dictionary and lists them on the terminal (or on the line printer if P is specified). If no file-name is specified, the account's master dictionary will be used. The general form is:

```
    LISTFILE {file-name} {(P)}
```

LISTPROCS PROC

The LISTPROCS PROC sorts all PROCs in any file or dictionary and lists them along with a brief abstract on the terminal (or the line printer if P is specified). If no file-name is specified, the account's master dictionary will be used. The general form is:

```
    LISTPROCS {file-name} {(P)}
```

LISTU PROC

The LISTU PROC lists the account name of all users currently active on the system, along with their logon time and channel number. The general form is:

```
    LISTU or LISTUSERS
```

LISTVERBS PROC

The LISTVERBS PROC sorts all verbs (not PROCs) in any dictionary and lists them on the terminal (or on the line printer if P is specified). If no file-name is specified, the account's master dictionary will be used. The general form is:

```
    LISTVERBS {file-name} {(P)}
```

2.11 BLOCKING PRINTING: BLOCK-PRINT

The BLOCK-PRINT command will print characters in a block-form on the line printer or the user's terminal, respectively. Any ASCII characters may be printed.

The general form of the BLOCK-PRINT command is as follows:

```
    BLOCK-PRINT character-string {(P)}
```

This command causes the specified character-string to be block-printed on the terminal. Any character-string containing single quotes (') must be enclosed in double quotes ("), and vice versa. The surrounding quotes will not be printed. A character-string not containing quotes as part of the string need not be surrounded by quotes.

The option "P" will route the output to the line printer.

Character-strings to be blocked cannot have more than eight characters (the total number of characters must not exceed the current line length set by the most recent TERM command).

If a BLOCK-PRINT command is illegally formed, any of the error messages 521 through 525 may be displayed.

2.12 COPYING DATA: COPY

The COPY processor allows the user to copy items from a file to the terminal, the line printer, to the same file, or to another file (either in his account or in some other user-account).

The COPY processor is invoked via the COPY verb, which is a Type II verb and therefore takes on the Type II format. The general form of the COPY command is:

```
COPY {DICT} file-name item-list {(options)}
```

The file-name parameter specifies the source file. The item-list specifies the items to be copied and consists of one or more item-ids separated by blanks, or an asterisk (*) specifying all items. The options parameter, if used, must be enclosed in parentheses. Options are described in the next section.

Once a COPY command has been issued, the COPY processor will respond in different ways depending on whether the copy is to the terminal, line printer, or to a file. Copy to the terminal may be specified by the presence of the "T" option, copy to the printer is specified by the "P" option. If neither of these options is specified, the copy is to a file.

On a terminal or line printer copy, the data is displayed in the following format:

```
-   item-id
001 attribute one
002 attribute two
003 attribute three
........
nnn last attribute
```

For example, the item "ITEMX" in the SAMPLE-FILE may be copied to the terminal as follows:

```
>COPY SAMPLE-FILE ITEMX (T) [CR]

    ITEMX
001 3745
002 SMITH, JOHN
003 1234 MAIN STREET
```

If the copy is a file-to-file copy, the processor will respond with:

```
TO:
```

The response to this request is in the form:

```
{(} {DICT} file-name {item-list}
```

where:

1. If the data is to be copied to a different file, the destination file-name is entered preceded by a left parentheses; the word DICT may optionally precede the file-name if the data is being copied to a destination dictionary file instead of a data file.
2. If the data is being copied to the same file, the left parentheses is omitted.
3. If the item-ids of the items being copied are to be changed, the list of new item-ids must follow.
4. If only a carriage return is entered to the "TO" request, a copy to the terminal is performed (just as if the original COPY statement had used the "T" option).

This is discussed further in the next sections.

2.12.1 FILE-TO-FILE COPY

Multiple items may be specified as the source and as the destination in the COPY statement. Multiple item-ids are separated by blanks, unless the item-id itself has embedded blanks, in which case the entire item-id must be enclosed in double quotes (" ").

For example, the item-list may be:

```
    1024-24 1024-25 "TEST ITEM" ABC
```

which specifies four item-ids, "1024-24," "1024-25," "TEST ITEM," and "ABC".

Item-ids may be repeated within the item-list. There may be different numbers of items within the source and destination lists. If the source item-list is exhausted first, the COPY terminates. If the destination item-list is exhausted first, the remainder of the items are copied with no change in item-id.

If the items are to be copied without any change in the item-ids, the destination file item-list may be null.

If it is desired to copy all existing items, an asterisk (*) may be used as the source file item-list.

If a preselected list of items is to be copied, the source item-list should be null; in this case, the COPY statement must be preceded by a SELECT, SSELECT, QSELECT, or GET-LIST statement.

When copying from one dictionary to another, the COPY processor does not copy dictionary items with D/CODE of "D" (i.e., the D-pointers). D-pointers must only be created by the CREATE-FILE processor. To recreate both the dictionary and the data sections of one file in a new file, a command sequence, such as shown in Example C on the following page, must be used.

2.12.1.1 Copy to Another Account

When copying to a file in another account, it is necessary to set up a Q-pointer to that account. Q-pointers, or File Synonym Definition items, are discussed in Section 3.11 of this manual.

Example A - Copying items to the same file:

```
>COPY DICT SAMPLE COST   (I) [CR] <------ Single dictionary
TO: PRICE  [CR]                           item copied

1 ITEMS COPIED

>COPY SAMPLE 1242-01  [CR]    <---------- Single data item
TO: 1242-99  [CR]                         copied

    1 1242-01     <---------------------- Item-id is listed
1 ITEMS COPIED

>COPY FLAVORS RED WHITE BLUE   [CR]    <--- Multiple data items
TO: ALPHA BETA GAMMA  [CR]                    copied

    1 RED
    2 WHITE
    3 BLUE

3 ITEMS COPIED
```

Example B - Copying items to a different file:

```
>COPY DICT SAMPLE * (I)   [CR]  <-------- All dictionary
TO: (DICT FLAVORS   [CR]                  items copied
[418] FILE DEFINITION ITEM 'SAMPLE' WAS NOT COPIED

2 ITEMS COPIED
```

Example C - Recreation of entire dictionary and data sections:

```
>CREATE-FILE (NEW-SAMPLE 1,1 3,1)  [CR]  <------ New file created

[417]  FILE 'NEW-SAMPLE' CREATED; BASE = 15417, MODULO = 1, SEPAR = 1
[417]  FILE 'NEW-SAMPLE' CREATED; BASE = 15418, MODULO = 3, SEPAR = 1

>COPY DICT SAMPLE * (I)  [CR]  <----------- All dictionary items
TO: (DICT NEW-SAMPLE)    [CR]               (except D-pointer) copied
[418]  FILE DEFINITION ITEM 'SAMPLE' WAS NOT COPIED

3 ITEMS COPIED
>COPY SAMPLE * (I)  [CR]  <---------------- All data items copied
TO: (NEW-SAMPLE     [CR]

22 ITEMS COPIED
```

2.12.1.2 COPY Options

This section describes the options that may be specified in the COPY statement.

The general form of the COPY command is:

```
    COPY {DICT} file-name item-list {(options)}
```

The "options" parameter, if used, must be enclosed in parentheses. Options are single alphabetic characters; multiple options may be strung together, or separated by commas for clarity.

Table 2-2 describes the options used by the COPY processor. Note that some options operate differently depending on whether the copy is to the terminal or line printer or is a file copy.

Table 2-2. COPY Processor Options

Option	Note*	Description
A		Activates assembly MLIST format.
D	1	Delete item. On a file copy, the original (source) item is deleted from the file after it is copied.
F	2	Form-feed. On a copy to the terminal or the line printer, item will cause a new page to begin.
I	1	Item-id list suppress. On a file copy, will inhibit the listing of item-ids copied.
M		Activates Macro (Assembly) format.
N	1	New item inhibit. On a file copy, will not copy the items to the destination file unless the item already exists there. New items will not be created if this option is set.
N	2	No page. On a terminal copy, will inhibit the automatic end-of-page wait.
O	1	Overwrite items. On a file copy, will copy the item to the destination file even if it already exists on file.
P		Printer copy. Copies the data to the SPOOLER.
S	1	Suppress error messages. On a file copy, messages indicating that items were not copied (messages 409, 415, and 418) will not be printed.
S	2	Suppress line numbers. On a copy to the terminal or line printer, the line numbers will not be displayed
T		Terminal copy. Copies the data to the terminal.
X	1	Hexadecimal format. On a terminal or line printer copy, the data is displayed in hexadecimal form.
n		An integer number indicating the number of items to be copied. Used for copying data for test files.

*NOTES

1. Valid only on a FILE copy.
2. Valid only on a NON-FILE (terminal or line printer) copy.

2.13 PROGRAM INTERRUPTION: DEBUG FACILITY

Processing can be interrupted by pressing the BREAK key on the terminal (INT key on some terminals). This causes an interrupt in the current processing and an entry into the DEBUG state. This is inhibited during critical stages of processing.

When the BREAK (or INT) key has been pressed and the DEBUG state has been entered, the following message will be displayed:

```
I x.d
!
```

where:

x and d	describe the software location of the interruption.
!	is the DEBUG prompt character and is displayed to prompt the user for a DEBUG command.

For users with system privilege levels 0 or 1, the commands listed below are the only DEBUG commands allowed. Users with system privilege level 2 should refer to the PICK Operator Guide for further DEBUG facilities. The user should note that pressing the BREAK key while in the terminal input or output mode will cause a loss of up to 16 characters. If in the input mode, the retype line character (control-R) should be used to check the loss of data after returning from DEBUG via the G command.

Command	Description
P	Print on/off. Each entry of a P command switches (toggles) from print suppression to print non-suppression. The message OFF is displayed if output is currently suppressed. The message ON is displayed if output is resumed. This feature is useful in limiting the output at the terminal.
G or g or LINE FEED	Go. This command causes resumption of process execution from the point of interruption. G or LINE FEED cannot be used if a process ABORT condition caused the entry to DEBUG.
END or end	Terminates current process and causes an immediate return to TCL or to the LOGON PROC if the RESTART option is in effect.
OFF or off	Terminates current process and causes the user to be logged off the system.

Upon encountering one of the hardware abnormal conditions, the system will automatically trap to the DEBUG state with a message indicating the nature and location of the abort. If the user has system privileges level 0 or 1, he must type END or OFF to exit from the DEBUG state.

file structure 3

3.1 THE FILE HIERARCHY

This section describes the nature of the files in the PICK Operating System. Throughout this section, the following terms will be used:

Name	Conventional Data Processing Name
Item	Record
Attribute	Field
Item-id	Record Key

Files are organized in a hierarchial structure, with files at each level pointing to multiple files at the next lower level. Four distinct file levels exist:

1. System Dictionary
2. User Master Dictionary
3. File Level Dictionary.
4. Data File

The hierarchial file structure is illustrated in Figure 3-1. The term "file" refers to a mechanism for maintaining a set of like items logically together. The data in a file may be accessed via the "dictionary" associated with it. A dictionary is like the "index" to a file. Since the dictionary itself is also a file, it contains items like a data file. The items in a dictionary serve to define lower level dictionaries or data files.

The system can contain any number of files. Files can contain any number of items and can automatically expand to any size. Items are variable length and can contain any number of fields and characters so long as the data in an item does not exceed a maximum of 32,267 bytes.

3.1.1 SYSTEM DICTIONARY (SYSTEM)

The highest level dictionary is called the System Dictionary (SYSTEM). The System Dictionary contains all legitimate user Logon names, along with associated passwords, security codes, restart and history update flags, and system privileges. The Logon names and related information are stored as items in the System Dictionary. These items function as pointers to the user´s Master Dictionary.

```
Level 0          System Dictionary            One per system

            +-----------------------------+
            |    Account names with       |
            |    passwords accounting     |
            |    information.             |
            +-----------------------------+
                          |
                          |
                          v
Level 1         Master Dictionary (MD)        One per account

            +-----------------------------+
            |    Vocabulary items;        |
            |    verbs, modifiers, etc.;  |
            |    file-names.              |
            +-----------------------------+
                          |
                          |
                          v
Level 2            File Dictionary            At least one per data
                                              file, or data files
            +------------------------------+
            |    Data definitions and      |
            |    inter-relationship        |
            |    definitions.              |
            +------------------------------+
                          |
                          |
                          v
Level 3               File Data
            +--------------------------------+
            |    Data items.                 |
            +--------------------------------+
```

Figure 3-1. The Four-Level File Hierarchy

3.1.2 USER MASTER DICTIONARIES (MDs)

The Master Dictionaries (MDs) comprise the next dictionary level. Each user's account will normally have a unique MD associated with it. The MD contains items which store the definitions of all user vocabulary (verbs, PROCs, etc.) and items which function as pointers to accessible files.

When an account is created, a standard set of MD vocabulary items are stored in the account's MD. A user may, however, create synonyms to, and abbreviated forms of, these standard vocabulary elements (since they are merely items within his MD file) by creating copies of the elements. The user can also add statements called PROCs to the prestored vocabulary.

The file pointers can reference any file or dictionary in the system (i.e., they are not restricted to files defined within the user's account alone).

3.1.3 FILE LEVEL DICTIONARIES

The File Level Dictionaries describe the structure of the data within the associated data files. They also contain pointers to the associated data level files. A file level dictionary may be shared by more than one data level file.

Some dictionaries do not have an associated data file; these are called "single level" files. Data in a single level file are stored within the dictionary itself.

3.1.4 DATA FILES

The Data Files contain the actual data stored in variable record/field length format. In addition to the normal record/field data structure, an attribute (field) can contain multiple values, and a value, in turn, can consist of multiple subvalues. Thus, data may be stored in a three-dimensional variable length format.

3.2 FILE ACCESS

A file is a logical structure which associates a set of items so that they can be accessed for both retrieval and update.

The file access system is designed to allow the access of a particular item (or a number of particular items) in a file, or to access consecutively all items in a file.

Items are individually variable in length. The maximum size of an item is 32,267 bytes. There is no limit to the number of items which may be contained in a file, nor any limit to the number of files in an account. Each item has a "name" which is called its item-id. An item-id is an item identifier (key or name) that must be unique to the file which contains it.

Items are stored in the file in a "pseudo-random" sequence; this sequence is determined by the result of a computational "hashing" (randomizing) technique employed by the system for storage and retrieval of data on disk. This technique uses the item-id and other predefined parameters for the file to produce the disk-address (Frame-Identifier or FID), which identifies the location of the item.

Items that are stored in a file may be accessed directly, using the item-id as the key, or sequentially in the pseudo-random sequence. If items are to be accessed in any sorted sequence, a preliminary pass through the file to generate the sort sequence is needed (see SORT and SSELECT functions in the ACCESS Manual). The result of the preliminary pass is a "list" of item-ids; this list may be saved for future use or used immediately to access the items in the file in the required sorted sequence (see SAVE-LIST and GET-LIST functions in the ACCESS Manual).

The direct file access technique, which uses the item-id to locate the item within the file, is an efficient method of locating data, and lends itself to the on-line nature of the PICK system. The system overhead required to access an item using this technique is essentially independent of the actual size of the file.

Special reserved characters are used as delimiters for storing data within an item. Attributes are separated by "attribute marks" ('\', hexadecimal value X'FE') which may be subdivided into values by "value marks" (']', hexadecimal value X'FD'); the values may, in turn, be subdivided into subvalues by "subvalue marks" ('\', hexadecimal value X'FC'). This structure allows each attribute (including values and subvalues) to be of a variable length. This structure is discussed further in Section 3.8.

3.3 FILE STRUCTURE: BASE, MODULO, AND SEPARATION

The physical boundaries of the random access file are defined by three parameters: the Base, Modulo, and Separation. The physical boundaries are stored in the associated dictionary of the File Definition Item. The item-id of this item is the file-name.

Files are defined at the time of creation by these three parameters:

Base	Is the physical disk address (Frame-Identifier or FID) of the start of a contiguous block of reserved disk space. Base is automatically selected by the system.
Modulo	Is the number of groups that the file space is logically divided into (sometimes called "buckets"). Modulo is selected by user.
Separation	Is the number of sequential frames per group. Separation is selected by user. Default separation is 1.

The numbers chosen for Modulo and Separation determine how efficient the file access method will be. An algorithm for optimum selection is presented in the next section.

The Base, Modulo, and Separation of the file are stored by the CREATE-FILE processor when the file is created. These parameters should never be altered in any way by the user!

Therefore, at the time of file creation, a contiguous block of disk space is reserved. The size of this contiguous block is Modulo*Separation and is called the "PRIMARY SPACE" allocated to the file. This does not, however, define the TOTAL space available for the file. As data is placed into each group, the group may overflow by linking on additional disk frames as needed. There is no theoretical limit to this growth, other than the physical limit of disk space available. In practice, however, a group should be kept as small as possible. This may be achieved by the optimum selection of the file's Modulo.

The following shows INVENTORY file's defined base, modulo and separation:

```
    Item "INVENTORY" in the MD:  (Dictionary level file)

        INVENTORY

    001 D
    002 17324 <-----(base)
    003 3 <---------(modulo)
    004 1 <---------(separation)
       .
       .
       .
             FID    "Primary" space allocated to the
                    INVENTORY dictionary file.

                    ---------------------
             17324  |                   |  1st group
                    ---------------------
             17325  |                   |  2nd group
                    ---------------------
             17326  |                   |  3rd group
                    ---------------------

    Item "INVENTORY" in the dictionary INVENTORY:   (Data level file)

        INVENTORY

    001 D
    002 17573 <-----(base)
    003 373 <-------(modulo)
    004 1 <---------(separation)
       .
       .
       .
             FID    "Primary" space allocated to the
                    INVENTORY data file.

                    ---------------------
             17573  |                   |  1st group
                    ---------------------
             17574  |                   |  2nd group
               .    ---------------------      .
               .              .                .
               .              .                .
             17945  ---------------------  Last group
                    |                   |
                    ---------------------
```

3.3.1 ITEM STORAGE AND THE HASHING ALGORITHM

The system employs a computational group hashing technique which uses the item-id and the file parameters defined at the time of file creation. This technique generates the disk address (FID) of the group in which the item is stored.

The hashing formula used by the system to store or retrieve items is similar to the BASIC program shown below:

```
        X = 0
        FOR J = 1 TO LEN(ITEMID)
            X = X*10 + SEQ(ITEMID[J,1])
        NEXT J
        GROUP = REM(X,MODULO)
        FID   = GROUP*SEPARATION + BASE
```

where:

ITEMID contains the sequence of characters in the item-id.
The LEN function returns the number of characters in the item-id.
The form ITEMID[J,1] extracts the j-th. character of the item-id.
The SEQ function converts the above character to binary for addition.
The REM function returns the remainder of the division of X by MODULO.
And FID is the resulting disk address where the item may be found.

The item-id is treated as a variable length string of binary bytes; these bytes are accumulated sequentially with each partial sum being multiplied by 10. Dividing this value by the positive integer Modulo yields an unsigned integer remainder within the range:

0 <= Remainder < MODULO

This is then the group number (i.e., 0, 1, 2, ..., up to Modulo - 1) where the item is to be stored. Multiplying by the Separation and adding the Base yields the actual FID of the first frame in the group.

After computing a FID to locate the specific group in which the item resides, each item's item-id in the group must be compared for a "match." The frames comprising a group are linked both forward and backward. This system facility makes the group appear as a physically sequential string, where items are stored one immediately after another. In fact, any portion of an item may spill across a physical frame boundary.

When a file is created, it is allocated a primary area of frames, the number of frames being: Modulo*Separation. Thus, this amount of contiguous disk space is permanently allocated to the file. As the file grows, individual groups may fill up. When this happens, an additional frame is added to the group from a pool of available space. This additional frame is linked into the group to increase the length of the logically sequential group. Additionally, if a delete or update causes the group to shrink, any unused frames outside the primary area are returned to the pool of available space.

3.4 SELECTING MODULO AND SEPARATION

Effective file accessing and efficient disk utilization depends on proper selection of modulo and separation.

"Modulo" is the number of groups in a file; "Separation" is the number of contiguous frames per group. When a file is created, the user specifies its Modulo and Separation. The frames allocated by the system (Modulo*Separation) are referred to as the "primary" file space. As data is placed into the file, any group may overflow by attaching frames from the available system space pool; this space is referred to as the "overflow" file space.

When an item is to be added to a file, its item-id is "hashed" or converted into a large number by a multiplication process. This number is then divided by the modulo of the file. The remainder of this division is the "group" into which the item will be placed. This method will produce an even distribution of items in each group.

A proper selection of the "modulo" parameter is essential to minimize this search time.

In the current file structure, a Separation parameter of more than 1 will be of use only if the average item size is greater than 1000 bytes, in which case, a few disk reads are avoided when searching for (reading) an item in the prime file space. In all other cases, particularly in a multi-user environment, the disk head will almost certainly have moved between the moment that a process requests one frame of a group and the next; therefore, whether the next linked frame of the group is contiguous (i.e., if the separation is greater than 1) or not makes a marginal difference.

The number of groups in a file directly affects the search and update time for an item in the group. The Modulo selection process will attempt to make the average GROUP length between 1 and 2 frames. However, if the item size is of the order of 250 bytes or greater, this rule must be modified; you should then try to minimize as far as possible the average number of frames in a group. Therefore, the average number of items in a group should be selected with the average item size in mind; the larger the item size, the smaller the number of items in a group.

The number of disk reads, which is the factor that causes the most degradation of overall system response, increases dramatically as the number of frames per group increases, due to the fact that on the average, one-half of the frames in a group have to be written back to the disk after an item update. In order to update an item in a group, the system must read every frame in the group, and write and verify one-half of them. With this in mind, Tables 3-1 and 3-2 should be used as a guide in selecting Modulo. Separation should be 1.

The discontinuities in the items/group columns are because the selection of the number is such that the bytes/group figures are close to integral multiples of frames (500, 1000, 1500, etc.). The last figure, 0.8 items/group, may be used for files with relatively few items that are very large, such as Assembly or BASIC program files. If the number of items in such a file is also very large, adjust the items/group figure upwards, since the lower figure will result in a lot of wasted disk space. Using the table, you can select an appropriate items/group value; knowing the expected number of items in the file then gives you the approximate Modulo. Note that Modulo must not be a multiple of 2 or 5, and should preferably be a prime number.

Table 3-1. Selecting Items/Group

If Average Item Size Is:		Then Average Items/Group Should Be:		And Average Bytes/Group Will Be:
20	X	22.0	=	440
35	X	13.0	=	455
50	X	9.0	=	450
75	X	12.0	=	900
100	X	9.0	=	900
125	X	7.5	=	937
150	X	6.0	=	900
175	X	8.0	=	1400
200	X	7.0	=	1400
250	X	5.8	=	1450
300	X	6.4	=	1920
350	X	5.5	=	1925
400	X	4.8	=	1920
500	X	3.8	=	1900
1000	X	3.0	=	3000
5000	X	0.8	=	4000

Table 3-2. Examples of Computing Modulo

Average Item Size	Approximate Number of Items		Items/Group (From Table 3-1)		Approximate Modulo
20	800	/	22.0	=	36
40	5000	/	11.0	=	454
210	1800	/	7.0	=	257
4000	230	/	1.0	=	230

3.5 CREATING FILES: CREATE-FILE AND CREATE-PFILE

The CREATE-FILE processor generates new files and dictionaries in the system. It creates file dictionaries by reserving disk space and inserting a "D" entry in the user's MD which points to the file-level dictionary, and creates data files by reserving disk space and placing a pointer to the space in the file level dictionary. CREATE-FILE will automatically locate and reserve a contiguous block of disk frames from the available space pool. The user need only specify values for the Modulo and the Separation of both the file dictionary and the data area. For a discussion of the values to use for Modulo and Separation, refer to Section 3.4, SELECTING MODULO AND SEPARATION.

There may not be a data file without a file level dictionary pointing to it. Therefore, the file-level dictionary must be created prior to or concurrently with the data file. The latter is the preferred method for creating files and this form of the CREATE-FILE command is shown below. This enables the creation of both the dictionary and the data area with one command. The general forms are:

```
    CREATE-FILE file-name m1{,s1}m2{,s2}

    CREATE-FILE dict-name,data-name m1{,s1}m2{,s2}
```

where:

file-name	is the name of the file
m1 and s1	are the Modulo and Separation values of the dictionary (DICT) portion
m2 and s2	are the Modulo and Separation of the data portion
dict-name	is the name of the dictionary. (To be used when dictionary is shared by multiple data files.)
data-name	is the data file name. (To be used if multiple data files will be pointed to by the file dictionary.)

With either form, a pointer to the data file is placed in the file-level dictionary.

Note that if s1 and s2 are not specified, separation will be 1. If m1 or m2 are even numbers, the system will make them odd by adding 1 to the module.

A file dictionary may be created without a data file by the command:

```
    CREATE-FILE DICT file-name m1{,s1}
```

The term 'DICT' with modulo m1 and separation s1 specifies creation of the dictionary only. A pointer to file-name will be placed in the user's MD. Note that a data area need not be reserved for a single-level dictionary file; the data will be stored in the dictionary. (This is usually done for files that contain PROCs.)

Once a DICT has been created, primary file space for the data section of the file can be reserved. The general form of the command is:

```
CREATE-FILE DATA dict-name{,data-name} m2{,s2}
```

The term ´DATA´ specifies creation of the data file, data-name, if the data file is unique to the file-level dictionary, or creation of the data file, data-name, under dictionary, dict-name, if multiple data files are desired. Modulo m2 and Separation s2 are specified for the data file. The pointer to the reserved space will be placed in the file-level dictionary. This form is used to create new data files pointed to by a shared dictionary by including the name of the data file, data-name.

Examples of CREATE-FILE usage:

```
>CREATE-FILE INVENTORY 3,1 373,1  [CR]
```

Creates a new file called "INVENTORY", with a DICTIONARY section with Modulo of 3 and Separation of 1, and a DATA section with a Modulo of 373 and a Separation of 1. An item called "INVENTORY" will be placed in the MD and a D-item called "INVENTORY" will be placed in the INVENTORY dictionary.

```
>CREATE-FILE DICT TEST/FILE 7,1  [CR]
```

Creates a single-level dictionary file called "TEST/FILE"; a D-item "TEST/FILE" will be placed in the MD and a D-item "TEST/FILE" will also be placed in the dictionary created.

```
>CREATE-FILE DICT DEPT 3,1  [CR]
```

Creates a single-level dictionary called "DEPT".

```
>CREATE-FILE DATA DEPT,ACCOUNTING 73,1  [CR]
```

Creates a new DATA section called "ACCOUNTING" for the dictionary DEPT; a D-item called "ACCOUNTING" will be placed in the DEPT dictionary. The data file created will have to be referenced as "DEPT,ACCOUNTING" since it shares a dictionary.

```
>CREATE-FILE DATA DEPT,MAINTENANCE 57,1  [CR]
```

Creates a new DATA section called "MAINTENANCE" for the dictionary DEPT. This data file will have to be referenced as "DEPT,MAINTENANCE".

If you wish to create a pointer-file or a BASIC program file, use the CREATE-PFILE verb, which has the same general form as the CREATE-FILE verb.

```
    CREATE-PFILE file-name ml{,sl}m2{,s2}
      or         dict-name,data-name ml{,sl}m2{,s2}
      or         DICT file-name ml{,sl}
```

CREATE-PFILE is used to create POINTER and BASIC files and dictionaries because it will automatically place a "DC" in attribute 1 of the dictionary.

It will creates a file for file-name specified and its associated dictionary. The modulo and separation values for the dictionary are given first (ml,sl) and the values for the file last (m2,s2). If s is not specified, s=1.

The form dict-name,data-name must be used if file-name describes one of multiple files using the same dictionary.

A dictionary may be created without a data file by using the DICT form shown.

3.5.1 THE SHARING OF DICTIONARIES

File-level dictionaries may define a unique data file or multiple data files. When a dictionary defines multiple data files it is said to be "shared" by those data files. The characteristics of the data in the data files are typically similar.

For example, there may be sets of data relating to the various departments in a corporation. For ease of maintenance, these sets of data may share a dictionary, since the dictionary items that describe the data are identical for each department. These dictionary items, used by the ACCESS processor, apply to all of the data files defined by that dictionary. This structure has the advantage of requiring only one set of dictionary items to be maintained for a set of similar files.

Any number of data files sharing a dictionary may be opened simultaneously. The general form for specification of a data file is:

```
    dict-name{,data-name}
```

The first parameter, dict-name, always specifies the file dictionary. The second parameter, data-name, specifies the data file and is required ONLY in the case that multiple data files are using a common dictionary. If only one data file is using a dictionary, and it has the same name as the dict-name, then the form:

```
    file-name
```

specifies the dictionary and the data file of the same name.

For example, the inventory file may be called:

```
    INVENTORY
```

which indicates both the file and its dictionary. However, the departmental data files are sharing a dictionary called "DEPT", which requires a further specification. For example:

```
    DEPT,ACCOUNTING
          or
    DEPT,MAINTENANCE
```

Note that the dictionary of a file contains a "D" item which defines the associated data file. If the dictionary is NOT shared, the item-id of this pointer (file-name) is the same as that of the dictionary; this is the default case. Therefore, for example, the INVENTORY dictionary will contain an item, also called "INVENTORY", which is the pointer to the associated INVENTORY data file. The DEPT dictionary, on the other hand, will contain as many "D" items as there are departments; the item-ids of these pointers may be the department names.

The statements required to create a shared dictionary structure are shown below:

To create the dictionary of the file:

```
    >CREATE-FILE DICT DEPT m1,{s1}
```

To create the data section for each data file:

```
    >CREATE-FILE DATA DEPT,ACCOUNTING m2,{s2}

    >CREATE-FILE DATA DEPT,MAINTENANCE m2,{s2}
```

```
                             User MD
                     +---------------------+
                     |        DEPT         |
                     +---------------------+
                                |
                                v
                         DEPT Dictionary
            +-------------------------------------+
            |                                     |
            |   ACCOUNTING          MAINTENANCE   |
            |                                     |
            +-------------------------------------+
                       |                    |
                       v                    v
            Datafile ACCOUNTING   Datafile MAINTENANCE

            +-----------------+   +-----------------+
            |      Data       |   |       Data      |
            +-----------------+   +-----------------+

   File-name:  DEPT,ACCOUNTING      DEPT,MAINTENANCE
```

3.6 CLEARING AND DELETING FILES: CLEAR-FILE AND DELETE-FILE

The CLEAR-FILE processor is used to clear (i.e., to empty) files. The DELETE-FILE processor is used to delete files.

The CLEAR-FILE processor clears the data from a file. It sets the file to the "empty" state by placing X'FF' in the first data position of each group of the file. Overflow frames that may be linked to the primary file space will be released to the system's additional space pool. Either the data section or the dictionary (DICT) section of a file may be cleared using the CLEAR-FILE command. If the dictionary section is cleared, and a corresponding data section exists (as implied by the presence of a file definition item in the dictionary), then it will be maintained in the dictionary. The BREAK key is inhibited during the DELETE process, but not the CLEAR process.

To clear the data section of a file, the following command is used:

```
CLEAR-FILE DATA file-name{,data-name}
```

In the case that the data file is unique to dictionary, file-name, the data file, file-name is cleared; in the case that data file, data-name is one of multiple data files under dictionary file-name, then data-name will be cleared.

To clear the dictionary section of a file, the following command is used:

```
CLEAR-FILE DICT file-name
```

The DELETE-FILE processor allows you to delete the whole file consisting of both dictionary and data files, a dictionary only (if the dictionary has no attached data file), a unique data file or any of multiple data files. A file-level dictionary which points to a data file cannot be deleted. All frames owned by the deleted file are returned to the available space pool.

To delete a file-level dictionary and all its attached data file(s), use the command:

```
DELETE-FILE file-name
```

To delete a file-level dictionary without an attached data file, use the command:

```
DELETE-FILE DICT file-name
```

In both cases, the file definition item (D-pointer) in the user's MD is deleted and the space owned by the deleted file is returned to the available space pool.

To delete the data file only, the following command is used:

```
    DELETE-FILE DATA file-name{data-name}
```

This will delete the pointer to the data file from the file-level dictionary and return the space owned by the data file to the available space pool. The parameter "data-name" is necessary to delete a file that shares a dictionary with other data files.

Files that are defined by file-synonym definitions in the user's MD cannot be specified in a DELETE-FILE command.

Examples of DELETE-FILE and CLEAR-FILE:

```
    >DELETE-FILE INVENTORY
```

Deletes the INVENTORY dictionary and all associated data files.

```
    >CLEAR-FILE DATA INVENTORY
```

Clears the data section of the INVENTORY file.

```
    >DELETE-FILE DICT TEST/FILE
```

Deletes the dictionary TEST/FILE. If there are any data sections associated with this dictionary (i.e., if there are any D-items in the dictionary), this command will not be executed.

```
    >CLEAR-FILE DICT TEST/FILE
```

Clears the dictionary of the TEST/FILE of all items except D-items and Q-pointers.

```
    >DELETE-FILE DATA DEPT,ACCOUNTING
```

Deletes the DATA section ACCOUNTING from the shared dictionary structure whose shared dictionary name is DEPT.

The CREATE-FILE, DELETE-FILE, and CLEAR-FILE verbs do not require a left parenthesis prior to the file name or the key words DICT or DATA. Q-pointers may be used to reference files which are to be cleared, but the D-pointer name is required if the file is to be deleted. The CREATE-FILE verb constructs a D-pointer. If a data file is being added to an existing dictionary, the pointer to the dictionary will be a D-pointer.

3.7 THE DICTIONARIES

A dictionary defines and describes data within its associated file. The following dictionary levels exist within the system:

- System Dictionary (one per system)
- User Master Dictionary (one per user account)
- File Level Dictionary (one per file or files)

Since the dictionary itself is also a file, it contains items like a data file. The items in a dictionary serve as the actual definitions for lower level dictionaries or data files.

The following types of items are stored in dictionaries:

- File Definition Items (file-names/pointers)
- File Synonym Definition Items (file-names/pointers)
- Attibute Definition Items (attribute names)

File Definition Items, also called "D" items, are used to define files. File definition items are in the Master Dictionary of the user in whose account the associated data file exists. A file is said to be defined from the dictionary which contains the "D" item that points to it or, more accurately, to its dictionary. "D" items in the System dictionary point to the Master Dictionaries of each user or account. "D" items in data dictionaries define and point to their associated data file(s). The item-ids of "D" items are the names of the files to which they point. "D" items are automatically created by the Create-File processor when a file is first created. ("D" items should never be created by using the Editor or copied from another dictionary.)

File Synonym Definition Items, also called "Q" pointers, are used to access a file in another user's account. They are also used to provide a more convenient name for files within the user's own account. For example, to make typing in the file name easier, a "Q" pointer with the name "INV" might be created to point to the user's "INVENTORY" file. The user creates "Q" pointers in his MD by using the Editor.

Attribute Definition Items, also called "A" items, define the attributes that go into making up an item in an ACCESS data file. They are created in the dictionary for that data file.

For ACCESS processors, "A" items may contain such additional information as:

- Conversion specifications which are used to perform table look-ups, masking functions, etc.

- Correlative specifications which are used to describe inter-file and intra-file data relationships.

- Justification (left or right) for output purposes.

A data file is referenced by its "file-name." The dictionary file which is associated with that data file is referenced by "DICT" followed by the data file-name. A dictionary file may have more than one data file associated with it.

3.8 ITEM STRUCTURE (PHYSICAL)

Data within an item is stored as attributes, values and subvalues, all of which provide variable length storage. This section describes the physical item format as stored on disk.

An item consists of one or more variable length attributes, separated by attribute marks. An attribute mark is a character with a value of X´FE´ (hexadecimal), which prints outs as ´^´. The first attribute in an item (attribute 0) is the item-id. The item-id is preceded by a four-character hexadecimal count field which specifies the total number of characters in the item including the count field itself. For example, consider the following stored item:

```
002EITEMX^LINE1^SMITH, JOHN^1234 MAIN STREET^_
```

Attribute 0 is the item-id "ITEMX." It is preceded by "002E" which specifies that there are X´002E´ (decimal 46) bytes in the item. Attribute 1 of the "ITEMX" is "LINE 1." Attribute 2 is "SMITH, JOHN." Attribute 3 is "1234 MAIN STREET." The item ends with a segment mark, which is X´FF´.

An attribute, in turn, may consist of any number of variable length values separated by value marks. A value mark has an eight-bit value of X´FD´, which prints as "]". Finally, a value may consist of any number of variable length subvalues (also known as secondary values) separated by secondary value marks. A secondary value mark has an eight-bit value of X´FE´, which prints as " \". For example, consider the following item:

```
ITEM-ID  ATT1         ATT3      ATT4     ATT6       ATT7     END OF ITEM
      |    |            |         |       |            |        |
    __|____|_      _____|_________|_  ____|___________ |___     |

003AITEMY^05^AAAAA^123]456]78910^A^B^5]88\99\77\55]4^XYZ^_<----

-+---        ----+-- ---+---------  T   -------+----
 |               |      |           |          |
COUNT-FIELD    ATT2   MULTIVALUES   ATT5   MULTI-SUBVALUES
(003A = 58 decimal)
```

The absence of an attribute value is specified by placing an attribute mark immediately following the attribute mark that indicates the end of the previous attribute (i.e., ´^^´). This maintains the correct attribute sequence. The "null" between two adjacent attribute marks may be thought of as representing the absent attribute. Depending on the data structure, two adjacent value marks may be permissible, indicating the absence of a multivalue. Two adjacent submultivalues are normally not permitted.

The mnemonics AM, VM, and SVM will be used hereafter to denote attribute mark, value mark, and secondary (or sub) value mark.

The basic purpose of the value mark is to separate the contents of each value within each attribute. The purpose of this subvalue mark is to separate multiple entries within a value, within an attribute.

It is therefore, effective to store transaction records relating to a single vendor within one item. Within a single attribute, the fields from different values are separated by value marks. Attributes used in this manner are referred to as multivalued. Continuing the chain, a value within an attribute may itself contain several values. These are called subvalues and represent multiple subrecords within a given transaction record, as in the case of a purchase order specifying several different parts. The individual records remain identifiable because of the matching numbers of the delimiting value marks. The ACCESS processor is able to generate reports from this storage structure.

3.9 ITEM STRUCTURE (LOGICAL)

While it is important to understand the physical item format, in normal system usage, items are always accessed at a more abstract or higher level. Files are identified by a file name. Within a file, items are referenced by their item-id. Attributes are referred to as lines (e.g., attribute 1 is called "line 1"). The following example shows a sample COPY operation where the item with the item-id ITEMX (in the file SAMPLE-FILE) is being copied to the terminal. The item has three attributes (lines) of sample data.

```
    >COPY SAMPLE-FILE ITEMX (T)  [CR]

          ITEMX <------------------------------ Item-id
    001   5207  <------------------------------ Attribute 1
    002   SMITH, JOHN <------------------------ Attribute 2
    003   1234 MAIN STREET <------------------- Attribute 3
```

Utility processors like COPY and the EDITOR deal at the file-item line level. They make no logical distinction in definition between various lines in an item, other than their implied line numbers.

The ACCESS processor, however, adds an additional dimension through the use of the dictionary. The dictionary informs ACCESS of the nature of the information stored for each of the attributes.

An item is similar to a "record" in computer terminology. It is more effective to think of and use an item as a group of related records, however. One tends to see a record as a collection of fields distributed horizontally, having meaning by virtue of their offsets from the initial byte of the record.

In the PICK system, a data-string has meaning by virtue of its attribute number. If you think of an item (record) as a vertical list of attributes (fields) with attribute 1 on the first line, attribute 2 on the second line, etc., you get a clearer picture of the system's storage structure.

The logical item format is identical for all processors. It is the responsibility of the user to ascertain the further qualifications, if any, of the various attributes.

For example, the item listing in the previous example is shown below as produced by the ACCESS LIST processor.

```
    >LIST SAMPLE-FILE "ITEMX" ATTRIBUTE-1 NAME ADDRESS  [CR]

       PAGE 1                          09:28:32  12 JAN 1983

       SAMPLE-FILE..ATTRIBUTE-1..NAME............ADDRESS.........

       ITEMX        04/03/82     SMITH, JOHN      1234 MAIN STREET
```

Here, the SAMPLE-FILE dictionary "defines" attribute 2 (line 2) as NAME and attribute 3 (line 3) as ADDRESS. This permits the user to reference his data symbolically (through dictionaries). However, the actual data stored on file is the same regardless of the processor accessing it.

Also note that the COPY of the item displays a value of 5207 for attribute 1 of the item, whereas the ACCESS listing displays it as "04/03/82", which is the same data after conversion using the standard system date code.

3.10 FILE DEFINITION ITEMS

File definition items are used to define lower level dictionary files or data files. File definition items are specified by "D", "DC", "DCX", "DCY", "DY", or "DX". "D" pointers or "DC" pointers are created automatically by the CREATE-FILE and CREATE-PFILE verbs.

At the System Dictionary level, file definition items are used to define the Accounting File and each user's MD. File definition items in the MD define the file level dictionaries, which in turn may contain one or more file definition items which define the associated data file(s). The item-id and each attribute of the file definition item contains required and optional information which describes and 'points to' the lower level dictionary file or data file:

Item-id	The item-id of a file definition item is the file name of the dictionary or data file being pointed to. If the item is pointing to a data level file, then the item-id must be the same as the name of the data level file.
Attribute 1	This is the D/CODE attribute. It must contain one of the following: D, DC, DX, DY, DCX, or DCY: where: D identifies the file as a lower level dictionary file or a data file. When the file is created, the CREATE-FILE processor will place a D in this attribute. One or two letters may follow the D to indicate: C the file contains binary data. Used only by the system POINTER-FILE. X do not save this file on filesave tapes. The file will not exist after a file restore. Y do not save the data in this file on filesave tapes. On a file restore, the file will be recreated in an empty state.
Attribute 2*	This is the F/BASE (file Base) attribute; it must contain the base FID (as a decimal number) of the defined file.
Attribute 3*	This is the F/MOD (file Modulo) attribute; it must contain the Modulo (as a decimal number) of the defined file.
Attribute 4*	This is the F/SEP (file Separation) attribute; it must contain the Separation (as a decimal number) of the defined file.
Attribute 5	This is the L/RET attribute. It may contain retrieval lock codes.
Attribute 6	This is the L/UPD attribute. It may contain update lock codes.

*WARNING: The user should never alter Attributes 2 through 4.

Attribute 7	Reserved.
Attribute 8	Reserved.
Attribute 9	This is the V/TYP attribute. It contains the justification code for the values in the attribute.
Attribute 10	This is the V/MAX attribute. It contains the maximum length for values in this attribute in decimal.
Attribute 11	Reserved.
Attribute 12	Reserved.
Attribute 13	This is the F/REALLOC attribute, which allows for the reallocation of the physical extents of a file during a system File-Restore process (see PICK Operator Guide). The format of this specification is as follows: (m,s) where m and s are decimal numbers specifying the new Modulo and Separation parameters of the file. Another way the physical extents of the file may be altered is to COPY the file to a new file.

Table 3-3 illustrates a sample file definition item that defines the file level dictionary for the INVENTORY data file. This item has an item-id of INVENTORY and is stored in the user's MD. It also shows the file definition item which defines the data area of the INVENTORY file. This item also has an item-id of INVENTORY, but it is stored in the dictionary level file and points to the data level file. The user should note that in a single level (dictionary) file, the file definition item (e.g., INVENTORY) may be absent from the file dictionary or it may be present and point to the dictionary itself.

Table 3-3. Sample File-Definition Items

(Item-id)	INVENTORY (MD)	INVENTORY (in DICT INVENTORY)
D/CODE	001 D	001 D
F/BASE	002 17324	002 17573
F/MOD	003 3	003 373
F/SEP	004 1	004 1
L/RET	005	005
L/UPD	006	006
	007	007
	008	008
V/TYP	009 L	009 R
V/MAX	010 10	010 7

Note that the item "INVENTORY" in the Master Dictionary has definitions relating to the items in the dictionary of the INVENTORY file (such as V/TYP of "L" and V/MAX of "10"; the item "INVENTORY" in the INVENTORY dictionary has definitions relating to the items in the DATA section, such as V/TYP of "R" and V/MAX of "7".

3.11 FILE SYNONYM DEFINITION ITEMS

File synonym definition items are used to allow access to files in another user's account, or to define a synonym name for a file in the same account. File synonym definition items are specified by "Q" and are referred to as "Q-items". The item-id and attributes of a file synonym definition item are as follows:

Item-id	The item-id of a file synonym definition item is the synonym name by which the defined file may be referenced.
Attribute 1	This is the D/CODE attribute. It must contain a "Q".
Attribute 2	This is the S/ACCOUNT attribute. It must contain the name of the account in which the actual file is to be found. (The account name is an entry in the SYSTEM dictionary). If this attribute is null, then the synonym and the actual file are defined in the same account.
Attribute 3	This is the S/FILE attribute. It must contain the item-id of the actual file definition item to which the synonym equates (i.e., the actual file-name). If this attribute is null, it means that the user's MD is the file for which a synonym is being defined.
Attribute 4	Not used.
Attribute 5	This is the L/RET attribute. It may contain retrieval lock codes.
Attribute 6	This is the L/UPD attribute. It may contain update lock codes.
Attribute 7	Reserved.
Attribute 8	Reserved.
Attribute 9	This is the V/TYP attribute. It contains the justification code.
Attribute 10	This is the V/MAX attribute. It contains the maximum length for values in the attribute in decimal.
Attribute 11	Reserved.
Attribute 12	Reserved.
Attribute 13	Reserved.

A synonym file definition item is required in order to access a file in another account. In addition, there are many cases where it is convenient to reference a file within the same account by more than one name. In this case also, a Q-item may be created with attribute 2 of the Q-item null. (This is a better method for creating a synonym file pointer than creating duplicate D-pointers.)

A Q-item to another user´s MD should have the user´s account name in attribute 2, and a null attribute 3.

Q-items are created by using the EDITOR. The items are edited into the MD using the form:

```
EDIT MD item-id
```

where item-id is the synonym name being defined.

There is also a standard PROC, called SET-FILE, that creates a temporary Q-item called QFILE, which may be used to set up a pointer quickly. (This PROC is described in the Utilities Manual.)

Table 3-4 illustrates a sample INVENTORY file synonym definition item which allows the user access to the file in the account named SMITH; the user can reference this file via the synonym file-name INV. It also shows sample Q-items that point to another user´s MD, and to a file within the same account.

Table 3-4. Sample Synonym File Definition Items in Account ´JONES´

(Item-id)	MD	INV	USER3	SAMPLE
D/CODE	001 Q	001 Q	001 Q	001 Q
S/ACCOUNT	002	002 SMITH	002 SMITH	002
S/FILE	003	003 INVENTORY	003	003 SAMPLE-FILE
F/SEP	004	004	004	004
	...	...	...	

These "Q" items are in the Master Dictionary of the JONES account. Item INV is a synonym pointer to the file INVENTORY, which is defined as a file in the Master Dictionary of the SMITH account.

Note that the item MD contains only ´001 Q´, and that Q-pointers to other MDs do not have the ´MD´ in 3. Item USER3 refers to the Master Dictionary of user "SMITH," since attribute 3 is null.

Note that null attributes may not be filled with blanks. Since a null input (carriage return or line feed only) will cause the Editor to exit the input environment, it is necessary to place a temporary character in the "null" attribute and replace that character with a null via the REPLACE command. (See the EDITOR Manual.)

Item SAMPLE is a synonym to the file SAMPLE-FILE, defined in the Master Dictionary of JONES, since attribute 2 is null.

An example using the EDITOR to create the Q-item INV shown on the previous page.

```
EDIT MD INV [CR]
NEW ITEM
TOP
.I [CR]
001 Q [CR]
002 SMITH [CR]
003 INVENTORY [CR]
004 [CR]
.FI [CR]
´INV´ FILED.
```

3.11.1 THE REFLEXIVE FORM OF THE Q-POINTER

If attributes 2 and 3 are null, the Q-pointer is a pointer to the file in which it is stored. This case has two applications. If you type ED MD MD, you will find that the MD item contains only a Q in attribute 1. This is sufficient and any other definition is less efficient. Specifically, the MD entry should not be a D-pointer. The same follows for MD or the account name entry.

The second use is in the definition of a dictionary-only file. If you want to reference the file without typing ´DICT´ each time, an entry with the same name as the D-pointer to the dictionary in the master dictionary is inserted in the file dictionary whose only content is a Q.

Some uses of Q as the only attribute is as follows:

In the master dictionary

MD	File reference to MD
001 Q	Reference back to ´where you are now´

In the dictionary of the file FILENAME

FILENAME	The name referenced by the name FILENAME in the master dictionary
001 Q	Reference back to the dictionary itself

The name of the Q-pointer is discarded as soon as the first D-pointer is encountered. That is, a reference to QFILENAME which points to the file FILENAME will look for the D-pointer FILENAME in the dictionary of FILENAME. It will not look for a pointer by the name of QFILENAME. A partial exception to this occurs when using the ACCESS processor, which will attempt to obtain the conversion, length, and justification from the Q-pointer. If the Q-pointer does not contain them, then the ACCESS processor will search the D-pointer for them. If the D-pointer does not contain them, then the conversion will default to null, the justification to ´L´, and the field length to 10 bytes. It is, therefore, possible to specify various formats for the item-id field for purposes of sorting and listing.

3.11.2 ACCOUNT SPECIFICATION

The second attribute in any Q-pointer references an account name. If attribute 2 is null, then the Q-pointer references a file in the account onto which you are logged. If attribute 2 is not null, the file-open processor will search the system dictionary for a definition of the account name. If the processor does not find a D-pointer in the system dictionary, the system will respond with the following error message:

```
    [201] 'Contents of attribute 2 of QFILENAME' IS NOT A FILE NAME.
```

It is possible to reference files in the account onto which you are logged by putting the name of the D-pointer to the account in attribute 2 of the Q-pointer definition. However, this will cause the system unnecessary work.

Reference to the Master Dictionary of another account is done with the name of the D-pointer to the account (account-name) in attribute 2 and a null attribute 3.

A file or an account may be protected from access. (See Operator Guide, Section 7.2, System Security.) If a user creates a Q-pointer to a file or an account which he is not authorized to access, the system will deny him that capability. An error message stating that the file or account is protected will be returned.

3.11.3 FILE SPECIFICATION

Attribute 3 contains the name of the file referenced by the Q-pointer. If attribute 3 is null, then the default is to the Master Dictionary specified by attribute 2. MD should not be placed in attribute 3, since MD is a Q-pointer. The following error message will result:

```
    [201] 'MD' IS NOT A FILE NAME.
```

In general, the file-name referenced in attribute 3 of the Q-pointer definition must be a D-pointer in the MD of the account referenced in attribute 2.

3.11.4 EXTENSIONS TO THE FILE-NAME REFERENCE

PICK has the ability to reference multiple data files from a single dictionary. These are pointed to by D-pointers in the dictionary of the file. The names of the D-pointers in the dictionary are the names of the various data files. In order to reference a specific data file, use the string:

```
file-name,data-name
```

The contents of attribute 3 of the Q-pointer definition may contain file-name, data-name. In this case, the Q-pointer will reference the data in data-name only, and will ignore the other data files referenced in the dictionary of file-name. The result is a considerable simplification of the BASIC programs and PROCs which reference the various data sets in a multiple data file structure.

The following Q-pointer will reference the data file SCREW-DRIVERS in the dictionary of HARDWARE in the account INVENTORY.

```
QFILENAME
001 Q
002 INVENTORY <----- (account-name)
003 HARDWARE,SCREW-DRIVERS <----- (file-name,data-name)
```

A file dictionary does not need to reference a data file with the same name as the dictionary. It may be convenient to create the file as a dictionary-only file, which contains a Q-pointer that points to itself. The dictionary may then be referenced without typing 'DICT', which is useful if there will be extensive development of data definition items. It is also convenient if there will be several data-level files, since each may be given a recognizable name.

3.12 ATTRIBUTE DEFINITION ITEMS

Attribute definition items define various attributes (lines or fields) in the data items for use by the ACCESS processors. (More extensive treatment of this topic will be found in the ACCESS Manual.) Attribute definition items are specified by "A" in attribute 1 and are called "A-items".

An attribute definition item defines the nature and/or format of the data in a specific attribute. Each attribute definition item has a value, called the Attribute Mark Count (AMC), which acts as a pointer to the data field (data item attribute) defined by it. The AMC is simply the attribute number referred to in the data item. An AMC of 5 means that the attribute definition item "defines" attribute 5 of data items. An attribute definition item defines the attribute specified (by the AMC) for all items in the related data file(s). Moreover, an attribute definition item provides a name for an attribute.

Attribute definition items are constructed as follows:

Item-id	The item-id is the name desired for the defined attribute. This name should be used in ACCESS input statements to reference the defined attribute.
Attribute 1	This is the D/CODE attribute. It may contain an "A" (attribute), "S" (synonym attribute), or "X" (skip this attribute now).
Attribute 2	This is the A/AMC attribute. It contains the AMC of the defined attribute (i.e., it specifies which attribute in data item(s) is being defined). An AMC of 0 may be used to reference the item-id. An AMC of 0, or a "fake" value higher than the actual number of attributes that exist in the file, may be used if the attribute definition item references data that is not actually stored on the file, but is computed. In addition, an AMC of 9998 is used to access the current item counter (item sequence number), and an AMC of 9999 to access the SIZE or count field of the item.
Attribute 3	This is the S/NAME attribute. It contains the name to be used as the heading for this attribute in ACCESS listings. If this field is null, the item-id will be used for the heading.
Attribute 4	This is the V/STRUC attribute. It contains the associative structure code. (Refer to the ACCESS Manual.)
Attribute 5	Unused.
Attribute 6	Unused.

Attribute 7	This is the V/CONV attribute. It contains the conversion specification which is used to convert from internal processing format to output format.
Attribute 8	This is the V/CORR attribute. It contains the correlative specification which is used to convert from the internal format to processing format.
Attribute 9	This is the V/TYP attribute. It defines the justification (left or right) for output. V/TYP defaults to left justification if it is null.
Attribute 10	This is the V/MAX attribute. It defines the number of spaces that will be reserved for values for the attribute in vertical listings. The entry is a decimal number. V/MAX defaults to 10 if it is null.

An example of attribute definition items which define different fields in the INVENTORY file are:

(Item-id)	Quantity	List-Price	Extended-Price
D/CODE	001 A	001 A	001 A
A/AMC	002 4	002 5	002 300 — high number for calculating value
V/TAG	003	003 LIST PRICE	003
V/STRUC	004	004	004
	005	005	005
	006	006	006
V/CONV	007	007 MR2$,	007 MR2$,
V/CORR	008	008	008 A;4*5
V/TYP	009 R	009 R	009 R
V/MAX	010 7	010 8	010 10

3.13 DICTIONARY ITEMS: A SUMMARY

This section presents a summary of the items used in the various dictionaries in the system.

File, File Synonym and Attribute Definition Items - The File Definition, File Synonym, Attribute Definition, and Attribute Synonym Definition items which may be used as dictionary entries are summarized in Table 3-5.

System Dictionary (SYSTEM) Items - There is one, and only one, System Dictionary for each system. The System Dictionary should contain only items with D/CODE = D, DX, DY, or Q, to represent user accounts or special system files. The Logon processor uses these "D-items" to verify users attempting to log on to the system. Only one "D-item" should be present for each account; if more than one user name is to be established for the same user account, the additional name(s) should be File Synonym Definition ("Q-items"). The meaning of attributes 5 through 8 in the System Dictionary is different for both "Q" and "D" type entries. Entries in this dictionary completely control the File-Save process, which saves the data base on a secondary storage medium (typically magnetic tape).

Master Dictionary Items - There is one Master Dictionary for each account. The MD, like any other dictionary or data file, is comprised of items. Items with D/CODE of "A" define the attribute formats for all dictionaries. The file defining items (D/CODE of "D") point to the various files existing in that account.

In addition to those elements in the MD which define files and attributes, there are items which define verbs, PROCs, and various ACCESS language elements. Each of these items has a coding structure which uniquely identifies it. Refer to the following sources:

- Introduction to PICK TCL and FILE STRUCTURE Manual - Section 3
- PROC Manual
- ACCESS Manual
- Operator Guide

Table 3-5. Summary of File and Attribute Definition Items

Attribute Number	Name	File Definition Item	Synonym Definition Item	Attribute Definition Item
0	Item Identification	Item-id		
1	D/CODE	D, DX, DY, DC, DCX, DCY	Q	A, S, X
2	F/BASE or S/ACCOUNT or A/AMC	Base FID of file	Account-name	AMC
3	F/MOD or S/FILE or S/NAME	Modulo of file	Synonym file-name	Tag or Heading
4	F/SEP or V/STRUC	Separation of file	Not used	Controlling/ Dependent codes
5	L/RET	Retrieval lock code(s)		Reserved
6	L/UPD	Update lock code(s)		Reserved
7	V/CONV	Reserved		Conversion specification
8	V/CORR	Reserved		Correlative specification
9	V/TYP	Justification code		
10	V/MAX	Maximum field length		
11		Reserved		
12		Reserved		
13	F/REALLOC	Reallocation Specification	Reserved	

3.14 VERB DEFINITION ITEMS IN THE MASTER DICTIONARY

Each verb definition resides as an item in the user's MD. The item-id (i.e., attribute 0) of a verb definition item is the verb name itself. The attributes used in a verb definition item are defined as follows:

Attribute Number	Description
0	This is the item-id, which is the name of the verb.
1	Must contain one of the following: P, PA, PY, or PZ. where: P identifies the MD item as a verb definition item. The letter following P is passed to the defined processor. A defines an ACCESS verb. Y defines a TCL-II verb. Z defines a TCL-I verb.
2	This attribute defines the processor entry point to which TCL passes control (i.e., the mode-id in hex). An ACCESS verb will have an entry of 35 A Type-II verb will have an entry of 2 A Type-I verb will have an entry of xxxx where: xmmm x is the entry point, mmm is the mode-id in hex.
3	Secondary transfer point. Use depends on attributes 1 & 2.
4	Tertiary transfer point. Use depends on attributes 1 & 2.
5	TCL-II parameter string. These parameters govern item's retrieval by TCL-II verbs to be passed to processor whose entry point is defined in attribute 3. Parameter may be any of the following: C - Copy item to a work area. F - Pick up file parameters only (ignore item-list). N - Okay if item is not on file. P - Print item-id if item-list is "*" (all items), (or if SELECT-ed item-list). S - Ignore the select-list; item-list is mandatory. U - Items will be updated by processor. Z - Final entry required on EOI.

WARNING: Do not change any of this data in existing verbs!

As an example of a verb definition item, the following item (stored as item 'LIST' in the user's MD) defines the ACCESS verb LIST:

<u>Item 'LIST' in MD</u>

```
001 PA
002 35
003 4D
```

The following verb definition item defines the TCL-II verb MLIST:

<u>Item 'MLIST' in MD</u>

```
001 PY
002 2
003 20
004
005 C
```

As a final example, the following verb definition item defines the TCL-I verb TIME:

<u>Item 'TIME' in MD</u>

```
001 PZ
002 3033
```

The user may create any number of synonyms for the verb definition items (and may even remove the predefined verb definition items), thereby creating his own vocabulary. Synonyms may be created by copying the verb definition item into another MD item with the desired synonym name as the item-id.

3.15 INITIAL SYSTEM FILES/DICTIONARIES

The files described below are initial system files and are used in the operation and maintenance of the system.

The System Programmer (SYSPROG) account and the system message (ERRMSG) account are the only accounts needed to maintain the system. The prototype MD (NEWAC) is defined in the SYSPROG account. The ERRMSG file is accessed by all users to obtain error and informative messages, while the NEWAC file is used to create MD's for new accounts. SYSPROG also contains the system-level PROCs which perform the File-Save and File-Restore functions, and initialize the Accounting History file when the system is set up.

The ERRMSG File - This dictionary-level file in the ERRMSG account contains error and informative system messages. Each account's MD must have an item called ERRMSG which points to this file in the ERRMSG account. (This is automatically created by the CREATE-ACCOUNT PROC.)

The SYSPROG-PL File - This dictionary-level file contains the System Maintenance PROCs. These PROCs can be used from the SYSPROG account. Refer to the PICK Operator Guide for a description of the entries in this account.

The NEWAC File - This dictionary is defined from the SYSPROG account and is a prototype MD. It is used as a model by the CREATE-FILE PROC to create a new user's MD. The NEWAC dictionary level file contains three data files: NEWAC, SYS1, and SYS2.

The Accounting History File - The ACC file contains system accounting history and currently active (logged-on) users. The format of LOGON/LOGOFF accounting history statistics is described in the TCL section. The Accounting History File should be cleared periodically to prevent overflow of the file.

The PROCLIB File - The PROCLIB file contains all common PROCs (e.g., LISTU, CT, etc.). Each MD will contain a pointer to PROCLIB and items that transfer control to the corresponding PROCs in PROCLIB. For further information, refer to the PROC Manual.

The BLOCK-CONVERT File - This file contains items which are used by the BLOCK-PRINT verb to convert characters to a block format.

3.16 SPECIAL FILES

3.16.1 POINTER-FILES

Every pointer file must contain a 'DC' in attribute 1 of its definition. It must be two-level, but it is convenient to make the data-level pointer in the dictionary a Q-pointer to itself. The name POINTER-FILE is reserved and known to the list handler. It is therefore possible, and may be convenient, to call the actual pointer-file(s) by names different from POINTER-FILE and construct POINTER-FILE as a Q-pointer to the pointer file of the moment. Any pointer file in the system may be referenced this way. It is also possible to define several pointer files within one account. Each pointer file may then be used for a particular group of tasks that are executed on the account.

3.16.2 BASIC PROGRAM FILES

The BASIC program file must have a dictionary level and one or more data-level files, and the master dictionary entry for the BASIC program file must contain a 'DC' in attribute 1. The source code must be in a data-level file, and the dictionary will contain pointers to executable object code. This means that only the source items are in the program file.

If there are multiple data files, and if there is a program with the same name in more than one of them, the last one compiled is the one which will be run.

The CATALOG verb will include the name of the program in the master dictionary, with a pointer to the file which contains the particular program.

The DECATALOG verb will delete the object code from the system regardless of whether the program has been CATALOGed, and delete the cataloged entry in the MD.

INDEX