USER'S MANUAL

ICON/PICK OPERATING SYSTEM

ICON®

ICON INTERNATIONAL

ICON/PICK
USER'S MANUAL

ICON®

This manual has been prepared by the Documentation Support Group of Icon International, Inc., P.O. Box, 340 Orem, Utah 84057-0340.

Forms for readers' comments have been provided at the back of this publication. Comments are welcomed and may be sent to the address on the comments form.

Icon International, Inc. reserves the right to make changes, without notice, to the specifications and materials contained herein, and shall not be responsible for any damages (including consequential) caused by reliance on the material as presented, including, but not limited to, typographical, arithmetic, and listing errors.

Revision A

Order No. 172-026-001 (Manual Assembly)
Order No. 171-010-001 (Pages Only)

WARNING

Trademarks

TABLE OF CONTENTS

SECTION 1

INTRODUCTION TO THE ICON/PICK OPERATING SYSTEM

ICON®

Chapter 1

INTRODUCTION

THE PICK SYSTEM

USER MANUAL

1.1 WHAT IS THE PICK COMPUTER SYSTEM?

The PICK System is a generalized data base management computer system. It is a complete system that provides multiple users with the capability to instantly update and/or retrieve information stored in the on-line data files. Users communicate with the system through local or remote terminals to access files that may be private, common, or security-controlled. Each terminal user's vocabulary can be individually tailored to specific application vocabularies.

The PICK System includes the powerful, yet simple-to-use ACCESS inquiry language, the PICK/BASIC and PROC high-level languages, file maintenance tools, an EDITOR, complete programming development facilities, and a host of other user amenities. PICK System runs in an on-line, multi-user environment with all system resources and data files being efficiently managed by a true Virtual Memory Operating System that provides users with unrivaled performance and reliability.

The PICK System is exceptional when measured from any angle: system capability multi-user performance, file management languages, ease of programming, data structure, and architectural features. The high performance and fast response of the PICK System are possible only through the use of a unique business-oriented, machine-independent assembly language which greatly reduces system overhead and program execution time.

The System Software includes:

- Virtual Memory Manager.
- Multi-user Operating System.
- Special Data Management Instructions.
- Input/Output Processors.
- ACCESS, PICK/BASIC, PROC, TCL Languages.
- Selectable/automatic report formatting.
- Dynamic file/memory management.
- Selectable levels of file/data security.

The unique file structure provides:

- Variable length files/records/fields.
- Multi-values (and subvalues) in a field.
- Efficient storage utilization.
- Fast accessibility to data items.
- Selectable degrees of data security.
- File size limited only by size of disc.
- Record size up to 32K bytes.

1.2 AN OVERVIEW OF PICK COMPUTER SYSTEM'S MAJOR FEATURES

The PICK System is a system specifically oriented to provide a vehicle for the implementation of cost-effective data base management. Data base management systems implemented in the PICK System afford two major benefits: 1) providing accurate and timely information to form the basis for significantly improving the decision-making process, and 2) substantially reducing the clerical and administrative effort associated with the collection, the storage, and dissemination of the information pertaining to an organization.

The PICK System is a very efficient and effective tool for on-line data base management. Pick has implemented a truly revolutionary on-line transaction processing system. Three major components of the system are especially important:

- The virtual memory operating system
- The software level architecture
- The terminal input/output routines

The virtual memory operating system which has long been used in larger computer systems had previously been impractical for mini-computers due to the large amount of overhead needed for the operating system itself. In the PICK System, the virtual memory operating system has been optimized and coded in a highly efficient machine-independent assembly language which executes many times faster than conventional languages. Thus the overhead time is no longer a serious problem on the smaller computers.

Most sophisticated computer operating systems require vast amounts of memory to support them. Systems consuming more than one hundred thousand bytes are common. Only a small amount of main memory is needed to run the PICK System. Everything else (system software, user software and data) is transferred automatically into main memory from the disc drive by the virtual memory operating system only when required.

Data in the PICK System is organized into 512-byte pages (frames) which are stored on the disc. As a frame is needed for processing, the operating system automatically determines if it is already in core memory. If it is not, the frame is automatically transferred from the disc unit (virtual memory) to core. Frames are written back onto the disc on a "least-recently-used" basis. The virtual memory feature of the PICK System allows the user to have access to a programming area not constrained by core memory, but as large as the entire available disc storage on the system.

The second important feature is the software level architecture of the machine itself. Pick Systems has implemented a machine architecture expressly designed and optimized for data base management. The architecture of the PICK System includes very powerful instructions expressly designed for character moves, searches, compares, and all supporting operations germane to managing variable length fields and records. This architecture was designed without the inevitable restrictions imposed by being tied to any one piece of hardware! It is truly a machine-independent approach.

The third major feature is the handling of Input/Output (I/O) communications with the on-line terminals. In any minicomputer on-line application, one of the main problems is that of managing the I/O from on-line interactive terminals. As these terminals increase in number, the load on the CPU becomes overwhelming and consequently the response to the terminals degrades dramatically. Pick has implemented the I/O processing of the on-line terminals with an overlapped buffering concept. This means that other program execution need not be held up waiting for terminal input/output to complete. As a result, the central processing unit is utilized more completely and a very large number of terminals may be connected to the Pick System before any significant degradation in response time is detected.

In summary, the PICK System encompasses the following extraordinary features:

- True data base management.
- Complete small business computer capabilities.
- Virtual Memory Operating System.
- Multi-user capabilities.
- On-line file update/retrieval.
- ACCESS retrieval language.
- Variable file/record/field lengths.
- Dynamic file/memory management.
- Automatic report formatting.
- Total data/system security.
- Fast terminal response.
- Line printer spooling.
- Special data management processors.
- High-speed generalized sort.
- Big computer performance on Minis, Micros and Mainframes.

The processors available on the Pick Computer System comprise the most extensive data base management software available on any minicomputer. An overview of some of the processors available to all terminal users is presented in this topic. All processors are described fully in the sections devoted to them.

The ACCESS Processor

ACCESS is a generalized information management and data retrieval language. A typical ACCESS inquiry consists of a relatively free-form sentence containing appropriate verbs, file names, data selection criteria, and control modifiers. ACCESS is a dictionary-driven language. The vocabulary used in composing an ACCESS input sentence is contained in several dictionaries. Each user's vocabulary can be individually tailored to his particular application terminology. ACCESS encompasses the following extended features;

- Logical English word order and syntax for user inputs.
- Automatic or user-specified output formatting.
- Sorting capabilities plus generation of statistical information.
- Relational and logical operations.
- Verbs such as: LIST, SORT, SELECT, COUNT, STAT, etc.

The PICK/BASIC Processor

PICK/BASIC is an exceptionally powerful yet simple and versatile programming language suitable for expressing a wide range of processing capabilities. PICK/BASIC is a language especially easy for the beginning programmer to master. PICK/BASIC is an extended version of Dartmouth BASIC which includes the following features:

- Flexibility in selecting meaningful variable names.
- Complex and multi-line statements.
- String handling with variable length strings.
- Integrated with Data Base file access and update capabilities.
- Fully structured programming support.
- Re-entrant and recursive abilities.

The PROC Processor

The PROC processor allows the user to prestore a complex sequence of operations which can then be invoked by a single word command. Any sequence of operations which can be executed from the terminal can also be prestored via the PROC processor. The PROC processor encompasses the following features.

- Argument passing.
- Interactive terminal prompting.
- Conditional and unconditional branching.
- Pattern matching.
- Free-field and fixed-field character moving.

The EDITOR Processor

The EDITOR permits on-line interactive modification of any item in the data base. The EDITOR may be used to create and/or modify PICK/BASIC programs, PROC's, assembly source, data files, and file dictionaries. The EDITOR uses the current line concept; that is, at any given time there is a current line that can be listed, altered, deleted, etc. The EDITOR includes the following features:

- Absolute and relative current line positioning.
- Merging of lines from terminal or from other file items.
- Character string locate and replace.
- Input/Output formatting.

The File Management Processors

The file management processors provide the capabilities for generating, managing, and manipulating files (or portions of files) within the Pick system. The file management processors include the CREATE-FILE processor, the CLEAR-FILE processor, the DELETE-FILE processor, the COPY processor, CREATE-ACCOUNT and DELETE-ACCOUNT.

The Utility Processors

Numerous utility processors are also included which provide an extensive complement of utility capabilities for the system.

Software Processor Usage

These and any other software processors may be used by any or all terminals simultaneously. Processing is invoked through appropriate verbs contained in each terminal user's Master Dictionary. User accessibility to these capabilities may be limited by controlling the verb selection available in specific user's Master Dictionaries.

1.4 OVERVIEW OF TCL

The Terminal Control Language (TCL) is the primary interface between the terminal user and the various PICK System processors.

Most processors are invoked directly from the Terminal Control Language by a single input statement, and return to TCL after completion of processing. TCL prompts the user by displaying a ">". This is referred to as the "TCL prompt character". Input statements are constructed by typing a character at a time from the terminal until the carriage return or line feed key is depressed, at which time the entire line is processed by TCL. The first word of an input statement must be a valid PICK "verb".

One of the powerful features of the PICK System is the ability to customize the vocabulary for each user. Since verbs reside in the individual user's Master Dictionary (MD), the vocabulary may be added to or deleted from without affecting the other users. In addition, an

unlimited number of synonyms may be created for each verb. The PICK System operates in what is known as an "Echo-Plex" environment. This means that each data character input by the terminal is sent to the computer and echoed back to the terminal before being displayed on the screen. The user is thus assured that if the data character displayed on the terminal is correct, the data character stored in the computer is correct.

In addition to the standard ASCII (96) character set recognized, special operations are performed when certain control characters are detected. All other control characters are deleted from the input line that is passed to lower level processors.

1.5 DATA BASE MANAGEMENT PROCESSORS

DATA BASE MANAGEMENT PROCESSORS

The data base management processors provide the capabilities for generating, managing, and manipulating files (or portions of files) within the PICK System. The data base management processors include the CREATE-FILE processor, the CLEAR-FILE processor, the DELETE-FILE processor, and the COPY processor.

The CREATE-FILE Processor

The CREATE-FILE processor is used to generate new dictionaries and/or data files. The processor creates file dictionary entries in the user's Master Dictionary (MD), and reserves disc space for the dictionary and data portion of the new file. The user need only specify the name of the file and value for the desired "modulo". The "modulo" parameter is selected to balance storage efficiency and accessing speed, based on the number of items in the file, the average item size, etc. The required file space is allocated from the available file space pool. Files may automatically grow beyond their initial size when the system automatically attaches additional "overflow" space from the available file space pool upon demand.

The CLEAR-FILE Processor

The CLEAR-FILE processor clears the data from a file. "Overflow" space that may be linked to the primary file space will also be released to the available file space pool. Either the data section or the dictionary section of a file may be cleared.

The DELETE-FILE Processor

The DELETE-FILE processor allows for the deletion of a file. All allocated file space is returned to the available file space pool. Either the data section or the dictionary section (or both) of the file may be deleted.

The COPY Processor

The COPY processor allows the user to copy an entire file (or selected items from the file) to the terminal, to the line printer, to the magnetic tape unit, to another file (either in the same account or in some other user-account), or to the same file under a different name (item-id).

1.6 AN OVERVIEW OF SYSTEM UTILITIES

The Pick Utility processors provide an extensive complement of utility capabilities for the system.

The Pick Computer System includes a very large number of utility processors. These processors provide such capabilities as:

- Magnetic tape unit functions
- Mathematical functions
- Line printer spooling control
- File save/restore functions
- File statistics
- Creation of user-accounts
- Setting of terminal characteristics
- Block printing
- Virtual memory dumping
- Inter-user message communications
- Bootstrapping and cold-start
- Systems accounting

1.7 AN OVERVIEW OF ACCESS

ACCESS is a user-oriented data retrieval language used for accessing files within the Pick Computer System.

ACCESS is a generalized information management and data retrieval language. A typical ACCESS inquiry consists of a relatively free-form sentence containing appropriate verbs, file names, data selection criteria, and control modifiers. Each user's vocabulary can be individually tailored to his particular application jargon.

ACCESS is a dictionary-driven language to the extent that the vocabulary used in composing an ACCESS sentence is contained in several dictionaries. Verbs and file names are located in each user's Master Dictionary (M/DICT). User-files consist of a data section and a dictionary section; the dictionary section contains a structural definition of the data section. ACCESS references the dictionary section for data attribute descriptions. These descriptions specify attribute fields, functional calculations, inter-file retrieval operations, display format, and more.

ACCESS provides the ability to selectively or conditionally retrieve information and also provides an automatic report generation capability. Output reports may appear on the terminal or be sent to the line printer and are automatically formatted according to the user's specifications by the Pick system. The output may be sorted into any sequence defined by the user, and encompasses the following extended features:

- Relatively free-form input of word order and syntax.
- Automatic or user-specified output report formats in either columnar or non-columnar form.
- Generalized data selection using logical and arithmetic relationships.
- Sorting capability on variable number of descending and/or ascending sort-keys.
- Generation and retention of multiple specially selected and/or sorted lists for use by subsequent processors.
- User ability to define variables derivable from the data in the object file and from other files, and to search, select, sort, total, output and break on the basis thereof.
- Selection of sub-records within items containing multiple unit records and sorts and outputs based on them.
- Generation of statistical information concerning files.
- Support of 11 digit signed arithmetic.

1.8 AN OVERVIEW OF PICK/BASIC

The PICK/BASIC Language is an extended version of Dartmouth BASIC, specifically designed for data base management processing on the PICK System.

PICK/BASIC is an extremely powerful yet versatile programming language suitable for expressing a wide range of problems. Developed at Dartmouth College in 1963, Dartmouth BASIC is a language especially easy for the beginning programmer to master. PICK/BASIC is an extended version of Dartmouth BASIC with the following features:

- Optional statement labels (statement numbers)
- Statement labels of any length
- Alphanumeric variable names of any length
- Multiple statements on one line
- Complex IF statements
- Multi-line IF statements
- Formatting and terminal cursor control
- String handling with variable length strings
- One and two dimensional arrays
- Magnetic tape input and output
- Decimal arithmetic with up to 14 digit precision
- ACCESS data conversion capabilities
- PICK file access and update capabilities
- Pattern matching
- Dynamic file arrays
- External subroutines

1.9 AN OVERVIEW OF THE EDITOR

The EDITOR is a PICK processor which permits on-line interactive modification of any item in the data base.

The Pick EDITOR may be used to create and/or modify PICK/BASIC programs, PROC's, assembly source, data files, and file dictionaries.

The EDITOR is entered by issuing the EDIT verb. The general command format is as follows:

```
EDIT file-name item-id
```

The item specified by "file-name" and "item-id" will be edited. If the specified item does not already exist on file, a new item will be created.

The EDITOR uses the current line concept; that is, at any given time there is a current line (i.e., attribute) that can be listed, altered, deleted, etc. The Pick EDITOR includes the following features:

- Two variable length temporary buffers
- Absolute and relative current line positioning
- Line number prompting on input
- Merging of lines from the same or other items
- Character string locate and replace
- Conditional and unconditional line deletion
- Input/Output formatting
- Prestoring of commands

EDITOR commands are one or two letter mnemonics. Command parameters follow the command mnemonic.

1.10 AN OVERVIEW OF PROC

An integral part of the PICK System is the ability to define stored procedures called PROC's.

The PROC processor allows the user to prestore a complex sequence of TCL operations (and associated processor operations) which can then be invoked by a single command. Any sequence of operations which can be executed by the Terminal Control Language (TCL) can also be prestored via the PROC processor. This prestored sequence of operations (called a PROC) is executed interpretively by the PROC processor and therefore requires no compilation phase.

The PROC processor encompasses the following features:

- Four variable length I/O buffers
- Argument passing
- Interactive terminal prompting
- Extended I/O and buffer control commands
- Conditional and unconditional branching
- Relational character testing
- Pattern matching
- Free-field and fixed-field character moving
- Optional command labels
- User-defined subroutine linkage
- Inter-Proc linkage

1.11 AN OVERVIEW OF THE PICK OPERATING SOFTWARE

Although the user need never be concerned with the architecture and instruction set of the Pick computer, the following section is provided for those readers who would like some information on Pick's unique structure.

In the early development of the PICK System, the task of creating an efficient, flexible business information system was given to a team of knowledgeable systems designers. At the time they began, the hardware selection had not yet occurred. While most people might consider this a handicap, it was in fact a most fortuitous situation. Not being constrained by the limits of any one type of hardware, the designers had the freedom to create a new language, an assembly language that was optimized for business data processing.

The power and flexibility in this assembly language is the strength of the current PICK System. The Pick instruction set has been specifically designed for character moves, searches, compares, and all supporting operations pertinent to managing variable length fields and records. The virtual memory is disc which is divided into 512-byte frames. The virtual memory addressing range is currently 12,192,320 frames, which is in excess of 6.4 billion bytes of data.

The Virtual Machine has 16 addressing registers and one extended accumulator for each terminal. A return stack accommodating up to eleven subroutine calls for each terminal is also provided. By indirect addressing through any one of the 16 registers, any byte in the virtual memory can be accessed. Relative addressing is also possible using an offset displacement plus one of the 16 registers to any bit, byte, word (16 bits), double word (32 bits), triple word (48 bits) or quadruple word (64 bits) in the entire virtual memory. This means fast response time and very high system throughput.

The PICK Instruction Set

The PICK System has an extensive instruction set. The main features include:

- Bit, byte, word, double-word and triple-word operations.
- Memory-to-memory operation using relative addressing on bytes, words, double-words, and triple-words.
- Bit operations permitting the setting, resetting, and branching on condition of a specific bit.
- Branch instructions which permit the comparison of two relative memory operands and branching as a result of the compare.
- Addressing register operations for incrementing, decrementing, saving, and restoring addressing registers.
- Byte string operations for the moving of arbitrarily long byte strings from one place to another.
- Byte string search instructions.
- Buffered Terminal Input/Output instructions.
- Handling of all data and program address references by the virtual memory operating system.
- Operations for the conversion of binary numbers to printable ASCII characters and vice versa.
- Arithmetic instructions for loading, storing, adding, subtracting, multiplying, and dividing the extended accumulator and a memory operand.
- Control instructions for branching, subroutine calls, and program linkage.
- Efficient stack operations for use by high level languages.

For further details regarding the PICK instruction set, refer to the PICK Assembly Manual.

1.12 SUMMARY OF PICK IMPLEMENTATIONS

Pick Operating Software is not new or untried. Its origins go back to the mid 1960's. It has been a commercial success since the early 1970's. In this time the concepts of user friendly on-line inter-action have been validated over and over again. The PICK System helps solve the biggest problem facing the expanded use of computers today. The creation of sufficient high-quality application software to support the new lower-cost hardware is a monumental task. By providing the best possible application software development environment coupled with intelligent data base management functions and a non-procedural ACCESS language report generator, the PICK System reduces these programming requirements.

In addition to the direct benefits of the Operating System, there are many tangible indirect advantages available to new users. The vast base of application programmers as well as the many vertical market packages available make finding application software easier.

MICRODATA 1600 (8-bit firmware machine)
INTERTECHNIQUE Multi-6 (8-bit firmware machine)
EVOLUTION 280 (8-bit firmware machine)
ULTIMATE Honeywell Level-6
ULTIMATE DEC LSI-11
ADP HEWLETT-PACKARD H-P 3000/Series 30
ADDS Mentor - Z8000 (16-bit microprocessor)
DATAMEDIA Motorola - M68000 (16-bit microprocessor)
C.D.I./IBM Series 1 (16-bit software machine)
ALTOS - I8086 (16-bit microprocessor)
GENERAL AUTOMATION Zebra - M68000 (16-bit microprocessor)
S.M.I./IBM 4300 (32-bit software machine)
PICK SYSTEMS IBM PC-XT (16-bit microprocessor)
SMI/IBM CS9000 - M68000 (16-bit microprocessor)
PERTEC Sabre - M68000 (16-bit microprocessor)
TAU - M68000 (16-bit microprocessor)
WICAT - M68000 (16-bit microprocessor)
CLIMAX - M68000 (16-bit microprocessor)
CIE 680 Series
FUJITSU - I8086 (16-bit microprocessor)
NIXDORF - 8090 VM

Summary of PICK SYSTEM Hardware Implementations.

1.13 A GLOSSARY OF PICK TERMS

A GLOSSARY OF PICK TERMS

The very nature of the PICK OPERATING SYSTEM presents certain terms and definitions which may be unfamiliar to conventional system users. Those terms and definitions, together with some more universally accepted acronyms and 'buzz' words, have been combined together in the following Glossary, to aid the first-time user in deciphering common terminology used in a PICK SYSTEM Environment.

ABS	ABSolute data image - generally taken to mean the Operating System (PICK) Modes which are loaded to a particular disk-drive area of frames.
AMC	Attribute Mark Count - a value found in a attribute defining item which contains the count (# of delimiters) of attribute marks, thereby specifying which attribute (field of data) in an item it refers to.
ATTRIBUTE	Each item is made up of a number of data fields or attributes. City, State and Zip would certainly be three attributes included in a Name and Address File.

BIT	BInary digiT - a unit of information equal to one binary decision. An eight BIT unit is called a byte. A character of data is represented in the computer by a byte (or 8 BITS).
BOOLEAN	Refers to a system of mathematical logic dealing with classes, propositions, on-off circuits, etc. Taken by programmers to mean, AND-OR-NOT-EXCEPT-IF..THEN, thereby allowing for logical decision making.
BYTE	A group of 8 bits usually processed together in parallel. A character of data is represented in the computer by a BYTE (8 bits).
CONTROL CHARACTERS	Normal keyboard letters, numbers or symbols which are entered while the "CONTROL" key is held down. They are not normally printable characters.
COMPILE	The process of turning user-written code (a PICK/BASIC program) into machine executable code which then has meaning to the computer. Source Code is COMPILED in order to execute it.
CONVERSIONS	Instructions may be stored in attribute 7 of attribute definition items. These CONVERSION instructions convert formats, (such as time, date, decimals, etc.) for the data that the attribute definition refers to. Internal format is converted to external format upon output and vice-versa.
CORRELATIVES	Instructions may be stored in attribute 8 of attribute definition items. Similar to conversions, they differ only in the times that their instructions are applied to the data. Both conversions and correlatives perform a number of tasks and greatly reduce programming requirements.
CPU	Central Processing Unit - generally refers to that electronic circuit board in the computer which contains the main storage (MOS memory), arithmetic unit, and special registers.
CRT	Cathode-Ray Tube - a terminal with a video screen, also called a VDT.
DEFAULT	The way processing will be done unless otherwise specified. A default value is a value that the computer will use (pre-programmed) in cases where user-defined parameters are prompted for and not supplied by the operator.

DELIMITER

Special Characters used to separate data. System delimiters separate sub-values, values, and attributes.

DICTIONARY

A PICK dictionary is a special type of file. Normally, a data file dictionary will contain two types of items. One type (called a D-pointer) contains information about the size and location of its associated data file on the disk. The other type of item is the attribute defining item and is used to define attributes in the data file associated with the dictionary.

EDITOR

The EDITOR processor permits on-line interactive modifications to any item in the data base. It is the normal input processor for writing procs, programs, system management and the like.

FILE

A file is a logical structure which associates a set of items. On a PICK system, files are organized into a hierarchical structure. There are four distinct levels of files, the SYSTEM DICTIONARY, a users MASTER DICTIONARY, FILE-LEVEL DICTIONARIES and the DATA FILES. A PICK system can contain any number of files, which contain any number of items, limited only by the size of the disk drive.

FRAME

Disc drive storage is divided into sections called FRAMES. Each FRAME is numbered giving the system direct access to that particular frame-id or FID. The physical size of a frame is machine dependent, the most common size being 512 or 1024 bytes per disk frame.

GROUP

The number of GROUPS in a file is the same as the MODULO for that file. As items are added to the file, additional overflow frames are linked on to the "primary frames" as needed. The size of each GROUP would then depend on how many overflow frames have been linked on to the primary frame of that GROUP.

HARDWARE

The physical part of the system which you can see and touch. The CPU, disk drives, tape drives, terminals and printers are examples of HARDWARE. A PICK/BASIC program is an example of SOFTWARE.

ITEM

A record made up of attributes. ITEMS make up a file. ITEMS are variable in length, the maximum size being 32,267 bytes. There is no limit to the number of items in a file, other than the size of the disk drive. The name of an item is called the "item-id". The item-id is unique to the file which contains that ITEM.

ITEM-ID

The name of an item in a file. An ITEM-ID may be any combination of numbers or letters, except system delimiters. If blanks are used in the ITEM-ID, then the ITEM-ID must be enclosed by quotation marks when accessed.

MD or M/DICT

MASTER DICTIONARY - each user-account on the system has a MASTER DICTIONARY associated with it. It is structurally similar to all other files on the system. Many things that a user enters at the TCL prompt are contained in that users MASTER DICTIONARY (such as verbs, procs, connectives, file-names, etc.). Upon creation, a standard set of vocabulary items are copied into that new account's MASTER DICTIONARY. Additional items may be added or deleted to customize that users account, as needed.

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.

MONITOR

The MONITOR is that part of the underlying system software which handles the operating systems interaction with peripheral devices. (Disk requests, Terminal I/O, etc.)

NULL

A lack of information as opposed to a zero or blank for the presence of no information. A blank or space which you get from the terminal space bar is not a null.

O/S

Operating System. The software that controls the carrying out of computer programs and other system functions (scheduling, I/O control, etc.).

POINTERS

POINTERS are items in dictionaries which serve a number of purposes. "D"-type POINTERS provide FID information to locate items in the data portion of the file. They reside in that files dictionary. "Q"-type POINTERS enable users to access files which are in another account. "Q"-type POINTERS are also used to shorten filenames (INV instead of INVENTORY or AH3 instead of ACCOUNT-HISTORY,MARCH).

PROC

PROC is short for stored procedure. PROC allows the user to prestore a complex series of operations which can be invoked by a single command. Anything which can be done at the TCL level, can be accomplished with a PROC.

SOFTWARE	Programs, routines, codes and other written information for use with computers, as distinguished from equipment, which is referred to as "HARDWARE". The PICK OPERATING SYSTEM is SOFTWARE.
STRING	A STRING is any succession of characters. They may be numbers, letters, blanks or other characters. The PICK SYSTEM treats most data simply as a certain sequence of symbols or "STRING".
TCL	Terminal Control Language processor. TCL is the primary interface between end-users and the computer. When the computer "prompt character" is displayed and is waiting for user input, this is commonly referred to as being "at TCL". The TCL processor works on one statement at a time. Each statement begins with a verb. Only one verb is allowed per statement.
VALUE (MULTI/SUB)	The contents of an attribute, if not null, is called its "VALUE". An attribute may contain more than one value. If it does, each of these values is called a "MULTI-VALUE". A multi-value, in turn, may contain more than one value. If it does, these values are called "SUB-VALUES".
VDT	Video Display Terminal. Same as a CRT.

SECTION 2

THE ICON/PICK FILE STRUCTURE

Chapter 2

FILE STRUCTURE

THE PICK SYSTEM

USER MANUAL

This section describes the hierarchical nature of the files in the the PICK System.

Throughout these sections the following terms will be used:

NAME	CONVENTIONAL NAME
Item	Record
Attribute	Field
Item-id	Record Key

Files are organized in a hierarchical structure, with files at each level pointing to multiple files at the next lower level. Four distinct file levels exist: System Dictionary, User Master Dictionary, File Level Dictionary, and Data File.

The term "file" as used in the context of this system refers to a mechanism for maintaining a set of like items logically together. The data in a file must 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 just as a data file does. The items in a dictionary then serve to define 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 it does not exceed a maximum of 32,267 bytes.

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, 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.

USER MASTER DICTIONARIES (MDs)

The Master Dictionaries (MDs) comprise the next dictionary level. Each user's account has a unique MD associated with it. The MD contains items which make up most of the users 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 create synonyms and abbreviated forms of these standard vocabulary elements (since they are merely items within his Master Dictionary file) by creating copies of the elements. The user can also add to the prestored vocabulary statements called PROCs.

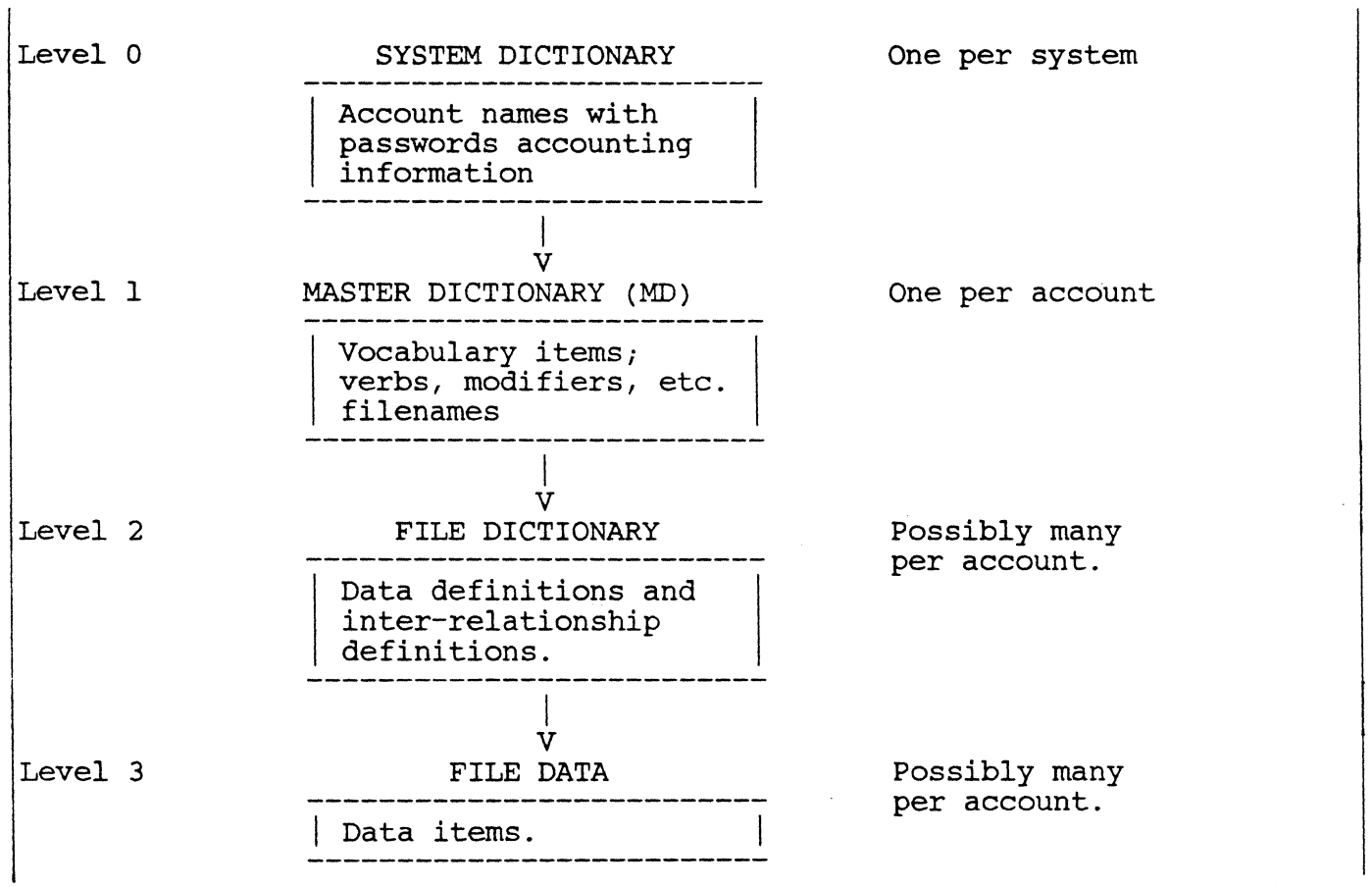
The file pointers can reference any file or dictionary in the system; that is, they are not restricted to files defined within the user's account alone.

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.

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 sub-values. Thus, data may be stored in a three dimensional variable length format.



The Four-level File Hierarchy.

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.

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

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 or key 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 which is employed by the system for purposes of storage and retrieval of data on disc. This technique utilizes the item-id along with other predefined parameters for the file, to produce the disc-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 ACCESS). The result of the preliminary pass is a list of item-ids; this list may be saved for future use, or used to then access the items in the file in the required sorted sequence (see also SAVE-LIST and GET-LIST functions in ACCESS).

The direct file access technique, using 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" ("^", control-shift-N on most terminals, hexadecimal value X'FE') which may be subdivided into Values by "Value-Marks" (']', control-shift-M, hexadecimal value X'FD'); the values may in turn be subdivided into Sub-values by "Sub-value-marks" ("\", hexadecimal value X'FC'). This structure allows each attribute (including values and sub-values) to be of a variable length. This structure is further discussed in the ITEM STRUCTURE, PHYSICAL section.

- THE SYSTEM CONTAINS ON-LINE:
  - ANY NUMBER OF FILES, WHICH CONTAIN:
    - ANY NUMBER OF ITEMS (RECORDS), WHICH CONTAIN:
      - MULTIPLE ATTRIBUTES (FIELDS), WHICH MAY CONTAIN:
        - MULTIPLE VALUES, WHICH MAY CONTAIN:
          - MULTIPLE SUB-VALUES.

- ALL FILES, ITEMS, ATTRIBUTES, VALUES, AND SUB-VALUES ARE VARIABLE IN LENGTH.
- EACH ITEM MUST BE LESS THAN OR EQUAL TO 32,267 CHARACTERS LONG

File Structure Summary.

2.3 THE DICTIONARIES THE DICTIONARIES

A dictionary defines and describes data within its associated file. Dictionaries exist at several levels within the system.

As introduced in the topic titled THE FILE HIERARCHY, 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 just as a data file does. The items in a dictionary serve as the actual definitions for data files. The following types of items are stored in dictionaries:

- File Definition Items (file-names/pointers)
  (also called "D"-items)
- File Synonym Definition Items (file-names/pointers)
  (also called "Q"-items)
- Attribute Definition Items (attribute names)
  (also called "A"-items)

The file definition items and the file synonym definition items are used to define files. The item-ids of these items are the file-names of the files they define or point to. File-names must start with a non-numeric character, and may be of any length and may contain any character except a comma (,) or a semi-colon (;). The attribute definition items are used to define attributes within data file items.

For example, "INVENTORY", "TEST.FILE" and "Z1" are all legal file-names. It is common practice to use file-names that are descriptive of the type of data stored within the file. A file is said to be defined from the dictionary that contains the "D-item" that points to it. Therefore, referring to the hierarchy of files in the system, all Master Dictionaries (or M/DICT's) are defined from the SYSTEM dictionary. In turn, a user may define any number of user dictionaries (with associated file or files) from his Master dictionary (see CREATE-FILE processor).

In order to access a file in another user's account, a file synonym definition item ("Q-item") may be created by the user, using the EDITOR. Assuming that the system security structure permits it, such a synonym file definition allows access to any file within the system.

A synonym file-pointer may also be used for convenience; for example the INVENTORY file may have a synonym file-name "INV", which reduces the number of characters the user has to type in order to access the file.

The data within each dictionary item consists of attributes (and optional multi-values) just as data file items.

For ACCESS processors, special dictionary items (called Attribute Definition items or A-items) define the nature of the data stored in their associated file. They 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.

- Type (alphabetic or numeric) and 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. This relationship is explained in the following section.

A dictionary contains:

1. File definitions, or "D-items", that define the physical extents of other, lower-level, files.

2. File synonym definitions, or "Q-items" that point to files in other accounts.

3. Data definition items "A-items" that are used by the ACCESS processor and define the structure of data in the data section of the file.

In addition, a Master Dictionary contains:

1. Verbs (see TCL documentation)

2. PROCs (see PROC documentation)

3. Vocabulary elements of the ACCESS language.

Summary of Dictionary Items.

2.4 SHARING OF DICTIONARIES

A dictionary file may be shared by any number of data files. This structure allows a unique set of dictionary items to define any number of like data files.

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, it may be desired to keep these sets of data in a shared dictionary structure, 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:

```
dictname{,dataname}
```


The first parameter, dictname, always specifies the file dictionary. The second parameter, dataname, 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 then the form:

filename

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

For example, the inventory file may be called:

INVENTORY

but the departmental data files, whose dictionary is called "DEPT", if using the shared dictionary structure, require a further specification. For example,

DEPT,ACCOUNTING or

DEPT,MAINTENANCE

As mentioned previously, 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.

Using the example below, the statements required to create a shared dictionary structure are:

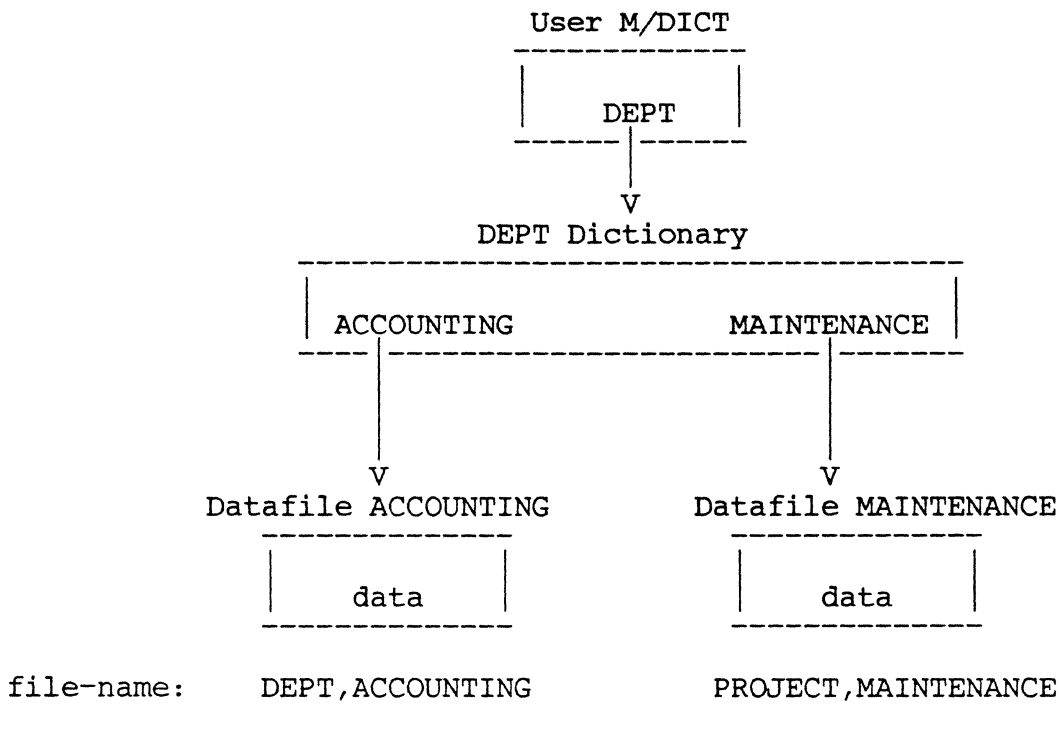
1. Create the dictionary of the file:

```
>CREATE-FILE DICT DEPT m  [CR]
```

2. For each data file, create the data section:

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

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

Example of the Shared Dictionary Concept.

2.5 BASE AND MODULO

The physical boundaries of the random-access file are defined by two parameters: the BASE and the MODULO

The physical boundaries of a file are stored (in the associated dictionary) in the File-Definition-Item. The item-id of this item is the File-name.

Files are defined at the time of creation by the following two parameters:

BASE — Is the physical disc address (frame-identifier or FID) of the start of a contiguous block of reserved disc space. This is automatically selected by the system.

MODULO — Is the number of groups that the file space is logically divided into (sometimes called "buckets"). (Selected by user.)

The selection of the MODULO is critical to the efficiency of the file access method. An algorithm for optimum selection is presented in the next section.

The BASE and MODULO 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 disc space is reserved. The size of this contiguous block is defined by the MODULO, 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 disc frames as needed. There is no theoretical limit to this growth, other than the physical limit of disc 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.

Item "INVENTORY" in the M/DICT:

```
    INVENTORY

001 D
002 17324
003 3
    .
    .
```

FID	"Primary" space allocated to the INVENTORY dictionary file.	
17324		1st group
17325		2nd group
17326		3rd group

Item "INVENTORY" in the dictionary INVENTORY

```
    INVENTORY

001 D
002 17573
003 373
    .
    .
    .
```

FID	"Primary" space allocated to the INVENTORY data file.	
17573		1st group
17574		2nd group

etc.

Example of a File's Defined BASE and MODULO.

Effective file accessing and efficient disc utilization depends on proper selection of the MODULO.

"Modulo" is the number of groups in a file. A file is created by specifying the "new-file-name" and a MODULO parameter; the frames allocated by the system

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. To locate an item given its item-id, the item-id is "hashed" using the MODULO of the file, which results in a unique number in which it may exist. The item-ids in that group are then linearly searched for the required item. A proper selection of the "MODULO" parameter is essential to minimize this search time.

Selecting a proper MODULO is extremely important, since the number of groups 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. Obviously, if the item-size is of the order of 250 bytes or greater, this rule must be modified; one 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 disc 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 disc after an item update. Thus to update an item in a group, we have to read every frame in the group, and write and verify one-half of them.

With this in mind, it is suggested that the tables below be used as a guide in selecting a proper MODULO.

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 disc space. Using the table, one can select an appropriate ITEMS/GROUP value; knowing the expected number of items in the file then gives the approximate MODULO.

MODULO MUST NOT BE A MULTIPLE OF 2 OR 5.

MODULO SHOULD BE A PRIME NUMBER.

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

Selecting Items/Group

Avg Item Size	Approximate # of Items		Items/Group (From Figure A)		Approximate Modulo
20	800	/	22.0	=	36
40	5000	/	11.0	=	454
210	1800	/	7.0	=	257
4000	230	/	1.0	=	230

Examples of Computing Modulo

2.7 ITEM STRUCTURE (PHYSICAL) ITEM STRUCTURE (PHYSICAL)

Data within an item are stored in terms of attributes, values and sub-values, all of which provide for variable length storage. This topic describes the physical item format as stored on disc.

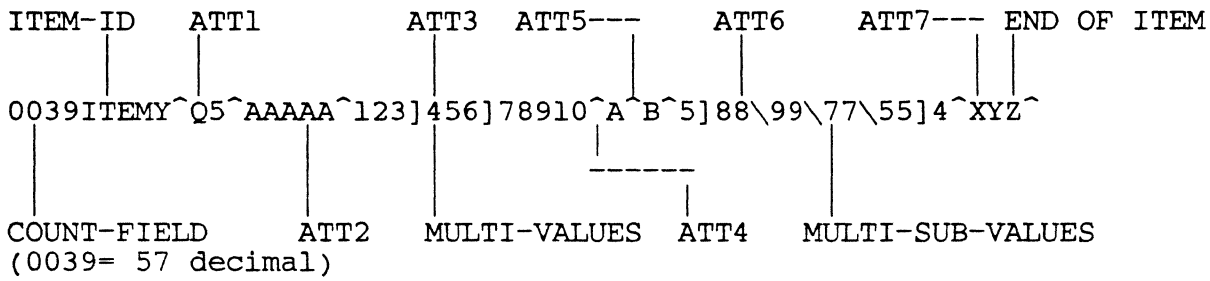
An item consists of one or more variable length attributes, separated by attribute-marks. An attribute mark is a character with a value of X'FE' (hexadecimal), which prints out as '^'. The first attribute in an item (attribute 0) is the item-id. The item-id is preceded by a four-character hexadecimal 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 "ITEMX" is "LINE 1". Attribute 2 is "SMITH, JOHN". The last attribute (attribute 3) is "1234 MAIN STREET".

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 sub-values (also known as secondary values) separated by sub-value marks. A sub-value mark has an eight bit value of X'FE', which prints as "\". For example, consider the following item:

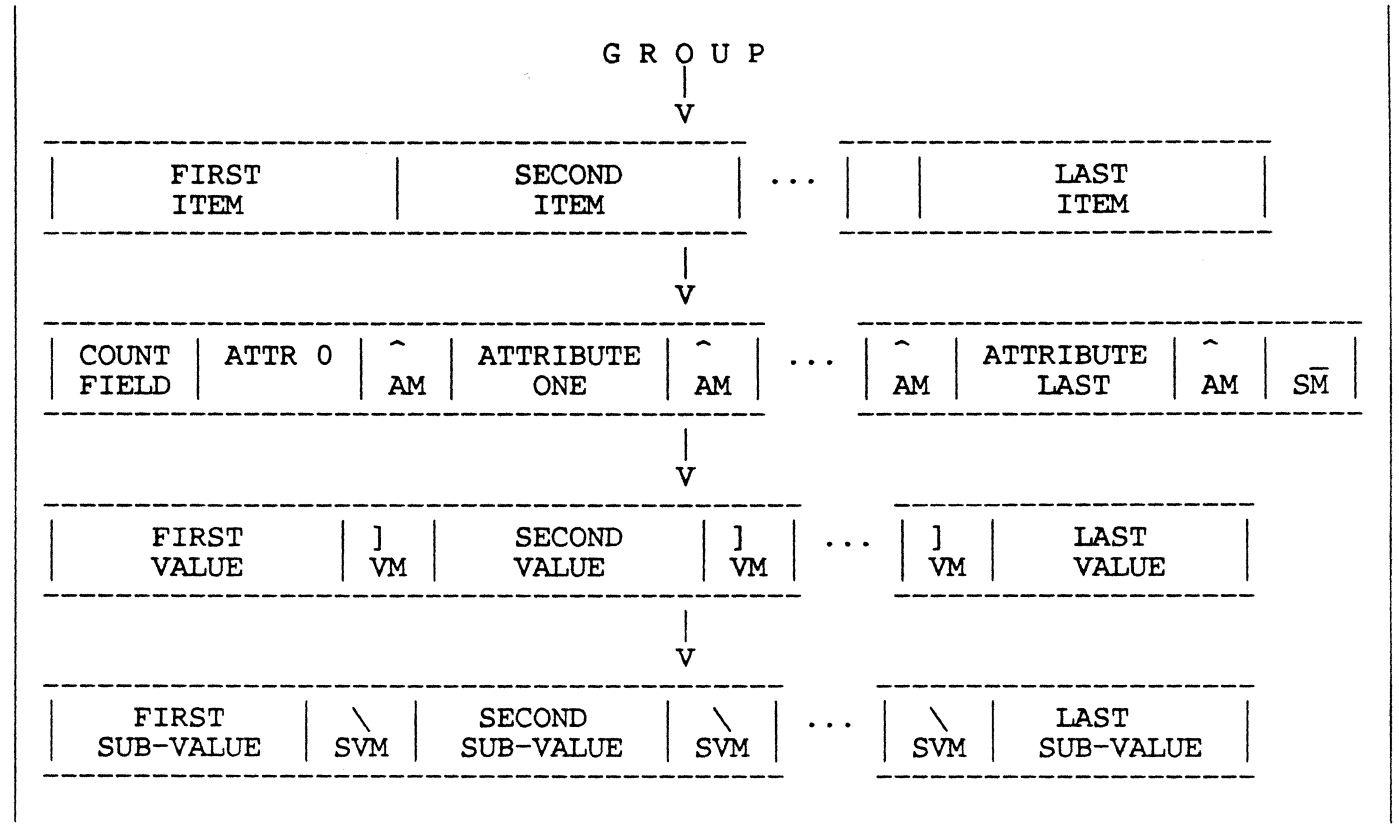


The absence of an attribute value is specified by an attribute mark immediately following the attribute mark indicating the end of the previous attribute (i.e. '^^'). This maintains the correct attribute sequence. The "null" between two adjacent attribute marks may be thought of as representing the absent attribute.

The mnemonics AM, VM and SVM will be used hereafter to denote attribute mark, value mark and sub-value mark.

Within a group, there may be zero or more items whose item-ids hash to that group. Such items are stored sequentially in the group, the sequence being solely dependent on the order in which the items are created.

After determining the group to which an item-id hashes, a linear search is conducted to find the particular item-id that is being retrieved. The count field is used to skip from one item to the next during this search. The presence of an SM where the count field of the next item should be indicates the END-OF-GROUP condition. An empty group therefore has an SM in the very first data position, which is also the condition setup by the CREATE-FILE and CLEAR-FILE processors.



General File Item Structure

2.8 ITEM STRUCTURE (LOGICAL) ITEM STRUCTURE (LOGICAL)

This topic describes the item structure at the logical level

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"). Figure A 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 is shown to have three attributes (lines) of sample data.

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.

ACCESS processors, however, add an additional dimension through the use of the dictionary. The dictionary informs ACCESS as to the nature of the information stored for each of the attributes.

It was noted that an Item is similar to a "record" in general parlance. It is more effective to think of and use an Item as a group of related records, however. In the general case one tends to see a unit record as a collection of fields distributed horizontally, and 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 the attribute (line) it is in. Therefore, if one thinks of a unit-record as running vertically down the horizontal attributes, such that the first field is in the first attribute, the second field is in the second attribute, and so on, the nature of the storage structure of the system becomes clearer. The basic intent of the value mark is to delimit the contents of each unit record within each attribute, and of the sub-value mark, to delimit multiple entries within a unit record within an attribute.

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

The logical item format is identical for all processors. It is the responsibility of the user to ascertain the further qualifications of the various attributes. In the examples below, the item listing in the first example is shown in the second example as produced by the ACCESS LIST processor. 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) when in fact 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 3746 for attribute 1 of the item, whereas the ACCESS listing displays it as "04/03/78", which is the same data after conversion using the standard system date code. (See ACCESS.)

```
>COPY SAMPLE-FILE ITEMX (T

       ITEMX  <-------------------------------- Item-id
001    3746 <---------------------------------- Attribute 1
002    SMITH, JOHN <--------------------------- Attribute 2
003    1234 MAIN STREET <---------------------- Attribute 3
```

An Item Listing Via the COPY Processor.

```
        X = 0
        FOR J = 1 TO LEN(ITEMID)
            X = X*10 + SEQ(ITEMID[J,1])
        NEXT J
        GROUP = REM(X,MODULO)
        FID   = GROUP + 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 disc address where the item may be found.

Hashing algorithm as expressed in PICK/BASIC terminology.

2.10 FILE DEFINITION ITEMS FILE DEFINITION ITEMS

File definition items are used to define lower level dictionary files or data files. File definition items are specified by a "D/CODE" of "D", "DY", or "DX". They are created automatically by the CREATE-FILE verb.

At the System Dictionary level, File Definition items are used to define the Accounting File and each user's MD (Master Dictionary). File definition items in the MD are used to 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 contain 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 a "D", followed optionally by a one or two character code.

When a file is created, the CREATE-FILE processor will place a "D" in this attribute. Alternate forms are:

Dx

x = X — Do not save this file on filesave tapes (the file will not exist after a file restore).

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

PAGE 1                                  09:28:32  12 JAN 1978

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

ITEMX          04/03/78      SMITH, JOHN       1234 MAIN STREET
```

An Item Listing Via the ACCESS LIST Processor.

2.9 ITEM STORAGE AND THE HASHING ALGORITHM

ITEM STORAGE AND THE HASHING ALGO

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

The hashing formula used by the system to store or retrieve items is shown below. 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. 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 physically frame boundary.

When a file is created, it is allocated a primary area of frames, the number of frames being the MODULO parameter. Thus this amount of contiguous disc 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. 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.

x = 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).

x = C The file contains binary data (presently used only by the system POINTER-FILE).

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.

WARNING: THE USER SHOULD NEVER ALTER ATTRIBUTES 2 through 4 !

Attributes 5 through 12 — These attributes are identical to those used in attribute definition items; refer to the topic entitled ATTRIBUTE DEFINITION ITEMS.

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 topic entitled SYSTEM MAINTENANCE PROCEDURES). 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.

The example below illustrates a sample file definition item which 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 is stored dictionary level file and points to the data level file.

(Itemid)	INVENTORY	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
	005	005
	006	006
V/CONV	007	007
	008	008
V/TYPE	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/TYPE 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/TYPE of "R" and V/MAX of "7".

Sample file-definition items.

File synonym definition items are used to allow access to files in another account, or to define a synonym to a file which is defined in the same account. File synonym definition items are specified by a D/CODE of "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 attribute must contain the name of the account in which the actual file definition is to be found (the account name is an entry in the SYSTEM dictionary). If this attribute is null, then the synonym file is defined in the same account.
Attribute 3	This is the S/NAME 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 is implied that the synonym file is the user's MD.
Attribute 4	Not used.
Attributes 5 through 10	These attributes are identical to those used in attribute definition items; refer to the topic entitled ATTRIBUTE DEFINITION ITEMS.

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 must be created; attribute 2 of the Q-item in this case should be NULL. A Q-item to another user's Master Dictionary should have the user's account-name in attribute 2, and a NULL attribute 3.

Q-items are created using the EDITOR to edit the items into the Master Dictionary. There is also a standard PROC called SET-FILE that creates a temporary Q-item called QFILE, which may be used to setup a pointer quickly. This PROC is described in the PROC reference manual.

The example 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 account's Master dictionary, and to a file within the same account.

(Item-id)	MD	INV	USER3	SAMPLE
D/CODE	001 Q	001 Q	001 Q	001 Q
F/BASE	002	002 SMITH	002 SMITH	002
S/NAME	003	003 INVENTORY	003	003 SAMPLE-FILE

These are example items in the Master dictionary of account "JONES"; Item "INV" is a synonym pointer to the file "INVENTORY", defined as a file in the Master dictionary of account "SMITH". Note that the form MD must be '001 Q', and that Q-ponters to other MDs do not have 'MD' in 3. Item "USER3" refers to file "USER3" in the Master dictionary of account "SMITH", since attribute 3 is null. Item "SAMPLE" is a synonym to the file "SAMPLE-FILE", defined in the Master dictionary of JONES, since attribute 2 is null.

Sample Synonym File Definition Items.

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

'INV' FILED.
```

NOTE: [CR] = press the carriage return key.

Example using the EDITOR to create a new Q-item called "INV".

2.11.1 Q-POINTERS : REFLEXIVE FORM

If attributes two and three 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 on an PICK System, you will find that the MD item contains only a Q in attribute 1. This is sufficient, and any other definition is less efficient. 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 contents is a Q.

```
In the master dictionary

    MD<File reference to MD.
001 QReference 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 QReference back to the dictionary itself.
```

Uses of Q as the only attribute.

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 is in ACCESS, 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 compiler 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 9 bytes. It is therefore possible to specify various formats for the item-id field for purposes of sorting and listing.

2.11.2 Q-POINTERS : 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 an error message.

Reference to the master dictionary of another account is done with the name of the D-pointer to the account in attribute 2 and a null attribute 3.

2.11.3 Q-POINTERS : FILE SPECIFICATION

Attribute 3 contains the name of the file referenced by the Q-pointer. If attribute 3 is null, then the default is the filename specified by the item-id of the item itself.

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

2.11.4 Q-POINTERS : MULTI-FILE SPECIFICATION

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

Therefore, the following Q-pointer will reference the data file DATAFILENAME in the dictionary of FILENAME in the account ACCOUNTNAME.

```
QFILENAME
001 Q
002 ACCOUNTNAME
003 FILENAME,DATAFILENAME
```

Referencing a data file with a Q-pointer.

2.12 ATTRIBUTE DEFINITION ITEMS

Attribute definition items define various attributes (lines or fields) in the data items for use by the ACCESS processors. Attribute definition items are specified by a D/CODE of "A".

An attribute definition item defines the nature and/or format of the data in a specific attribute for ACCESS processing. 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 (e.g. 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 symbolic name for an attribute.

Attribute definition items are constructed as follows:

Item-id	The item-id is the symbolic name desired for the defined attribute. This name would be used in ACCESS statements to reference the defined attribute.
Attribute 1	This is the D/CODE attribute; it may contain an "A" or "X".
Attribute 2	This is the A/AMC (attribute mark count) 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 ZERO may be used to reference the item-id. An AMC of zero, 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.
Attribute 3	This is the V/TAG attribute; it contains the optional name used as heading in ACCESS listings.
Attribute 4	This is the V/STRUC attribute; it contains the associative structure code (refer to the ACCESS reference 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 processing format to output format.
Attribute 8	This is the V/CORR attributes it contains the correlative specification which is used to convert from the internal format to processing format.
Attribute 9	This is the V/TYPE attribute; it defines the type (alphabetic or numeric) and justification (left or right) for output.
Attribute 10	This is the V/MAX Attribute; it defines the maximum length of values for the attribute. An entry is a decimal numeric, and is mandatory.

The example illustrates sample attributes definition items which defines different fields in the INVENTORY file.

(Item-id)	QUANTITY	LIST-PRICE	EXTENDED-PRICE
D/CODE	001 A	001 A	001 A
A/AMC	002 4	002 5	002 300
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/TYPE	009 R	009 R	009 R
V/MAX	010 7	010 8	010 10

Sample Attribute Definition Items in the Dictionary
of the Inventory File.

2.13 DICTIONARY ITEMS: A SUMMARY

DICTIONARY ITEMS: A SUMMARY

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

FILE AND ATTRIBUTE DEFINITION ITEMS

The File Definition items, File Synonym items, Attribute Definition items, and Attribute Synonym Definition items which may be used as dictionary entries are summarized below.

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, representing user accounts or special system files. The Logon processor uses these "D" type items to verify users attempting to logon to the system. Only one "D" type 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" type) items. The meaning of Attributes five through eight is different for both "Q" and "D" type entries in the System Dictionary. Entries in this dictionary completely control the File-Save process, whereby the data base is saved on a secondary storage medium (typically magnetic tape).

MASTER DICTIONARY (MD) 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 chapters for their respective definitions:

- TCL
- PROC
- ACCESS

ATTRIBUTE NUMBER	NAME	FILE DEFINITION ITEM	FILE SYNONYM DEFINITION ITEM	ATTRIBUTE DEFINITION ITEM
1	D/CODE	D, DX, DY, DC, DCX,DCY	Q	A,X
2	F/BASE or A/AMC	Base FID of file	Account-name	amc
3	F/MOD or V/TAG	Modulo of file	Synonym file-name	tag or heading
4	F/SEP or V/STRUC	Separation of file	Not used	C/D structure codes
5	L/RET	Retrieval lock code(s)		Reserved
6	L/UPD	Update lock code(s)		Reserved
7	V/CONV	Conversion specification(s)		
8	V/CORR	Reserved		Correlative
9	V/TYPE	Justification on type code		
10	V/MAX	Maximum field length		
11	. . . . .	Reserved		
12	. . . . .	Reserved		
13	F/REALLOC	Reallocation Specifica-tion	Reserved	

Summary of File and Attribute Definition Items.

2.14 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 is the only account needed to maintain the system. The system message file (ERRMSG) and the prototype MD (NEWAC) are defined in this account; the former is accessed by all users to obtain error and informative messages, while the latter is used to create new accounts' MDs. SYSPROG also contains the system-level PROCs which perform the File-Save and File-Restore functions, and the initialization of the Accounting History file on a System Setup.

THE ERRMSG FILE

This dictionary level file in the SYSPROG account contains the system messages (error and informative, see appendix). Each accounts' MD must have an item call ERRMSG which points to this file in the SYSPROG 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 topic entitled SYSTEM MAINTENANCE PROCEDURES 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 that is used as a model from which a new user's MD is created by the CREATE-ACCOUNT PROC.

THE ACCOUNTING HISTORY FILE

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

THE PROCLIB FILE

The PROCLIB file is used to contain 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 Reference Manual.

THE BLOCK-CONVERT FILE

This file contains items which are used by the BLOCK-TERM and BLOCK-PRINT verbs to convert characters to a block format.

THE POINTER FILE

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 or files by names different than POINTER-FILE, and construct POINTER-FILE as a Q-pointer to the pointer-file which is desired at 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, with the intent of using each pointer file for a particular group of tasks which may be executed on the account.

The pointer-file processor may reference only one pointer file at a time, however, and all processes logged onto a particular account will reference the same pointer-file.

THE PICK/BASIC PROGRAM FILES

The PICK/BASIC program file must have a dictionary level and one or more data-level files, and the master dictionary entry for the PICK/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. 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 now has the effect of including the name of the program in the master dictionary, with a pointer to the file which contains the particular program.

The DECATALOG verb is available to delete the object code from the system. It does not require that the program has been CATALOGed.

2.15 OVERVIEW OF FILE MANAGEMENT PROCESSORS

OVERVIEW OF FILE MANAGEMENT PRO

| This section describes the data base management processors for the system. |

The File Management processors provide capabilities for generating, managing, and manipulating files and items within the system. The File Management processors include the CREATE-FILE processor, the CLEAR-FILE processor and the DELETE-FILE processor.

Additional file management procedures (such as the creation of new user accounts, the saving and restoring of files, etc.) are detailed in the section entitled SYSTEM MAINTENANCE PROCEDURES.

THE CREATE-FILE PROCESSOR

The CREATE-FILE processor is used to generate new dictionaries and/or data files. The processor creates the file dictionaries which exist as the "D" entries (pointers) in the user's Master Dictionary (MD). The processor reserves and links primary file space. The user need only specify values for the desired modulo (number of groups in the file).

THE CLEAR-FILE PROCESSOR

The CLEAR-FILE processor clears the data from a file (i.e., it sets the file to the "empty" state by placing an attribute mark in the first data position of each group of the file). "Overflow" frames that may be linked to the primary frame space of the file will be released to the system's overflow space pool. Either the data section or the dictionary section of a file may be cleared.

THE DELETE-FILE PROCESSOR

The DELETE-FILE processor allows for the deletion of a file. Either the data section or the dictionary section (or both) of the file may be deleted.

If the file level dictionary is shared by several data files, each data file can be created, cleared or deleted independently of the other data files associated with the dictionary.

2.16 CREATING NEW FILES: THE CREATE-FILE PROCESSOR

The CREATE-FILE processor provides the capability for generating new files and dictionaries in the system.

The CREATE-FILE processor is used to create file dictionaries by reserving disc space and inserting a "D" entry in the user's Master Dictionary (MD) which points to the file-level dictionary, and to create data files by reserving disc space and placing a pointer to the space in the file level dictionary. CREATE-FILE will automatically locate and reserve a contiguous block of disc frames from the available space pool. The user need only specify a value for the modulo for both the file dictionary and the data area. For a discussion of the values to use for modulo, refer to the topic in this section entitled SELECTION OF MODULO.

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 a data area with one command. The general forms are:

```
CREATE-FILE filename m1 m2
CREATE-FILE dictname,dataname m1 m2
```


where "filename" is the name of the file, m1 is the modulo of the dictionary (DICT) portion, and m2 is the modulo of the data portion. Dataname is an optional data file name to be used if multiple data files will be pointed to by the file dictionary. In either case a pointer to the data file is placed in the file-level dictionary.

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

```
CREATE-FILE DICT filename m1
```

The term 'DICT' specifies creation of the dictionary only with modulo m1, and a pointer to filename is placed in the account's MD. The user should note that a data area need not be reserved for a single-level file, in which case the data are to be stored in the dictionary, as in the case of PROCS.

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

```
CREATE-FILE DATA dictname{,dataname} m2
```

where the term 'DATA' specifies creation of the data file dataname, if the data file is unique to the file-level dictionary, or creation of the data file dataname under dictionary dictname, if the multiple data file option is desired. The data file has modulo m2 and the pointer to the reserved space is placed in the file-level dictionary. This form is also used to create new data files pointed to by a shared dictionary using the option {dataname}.

* >CREATE-FILE INVENTORY 3 373 [CR]

Creates a new file called "INVENTORY", with a DICTIONARY section with modulo of 3, and a DATA section with a modulo of 373. 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 [CR]

Creates a single-level file called "TEST/FILE"; a D-item "TEST/FILE" will be placed in the Master dictionary, and a D-item "TEST/FILE" will also be placed in the dictionary created, pointing back to itself.

* >CREATE-FILE DICT DEPT 3 [CR]

Creates a single-level dictionary called "DEPT".

* >CREATE-FILE DATA DEPT,ACCOUNTING 73 [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 has the shared dictionary structure.

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

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

Examples of CREATE-FILE usage.

NOTE:

If you wish to create a pointer-file or a basic program file, use the CREATE-FILE verb, and then use the EDITor to change the D-pointer in the master dictionary to a DC-pointer.

2.17 CLEAR-FILE PROCESSOR CLEAR-FILE PROCESSOR

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

The CLEAR-FILE processor clears the data from a file (i.e., it sets the file to the "empty" state by placing an attribute mark 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 defining 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 filename{,dataname}

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

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

CLEAR-FILE DICT filename

>CLEAR-FILE DATA INVENTORY [CR]

Clears the data section of the INVENTORY file.

>CLEAR-FILE DICT TEST/FILE [CR]

Clears the dictionary of the TEST/FILE of all non-D-items; all D-ITEMS ARE MAINTAINED in the dictionary.

>CLEAR-FILE DATA DEPT,ACCOUNTING [CR]

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

Examples of CLEAR-FILE usage.

2.18 DELETE-FILE PROCESSOR DELETE-FILE PROCESSOR

The DELETE-FILE processor is used to delete files.

The DELETE-FILE processor allows the deletion of the whole file, dictionary and data files, the dictionary only (if the dictionary has no attached data file), the data file in the case of a unique data file, or any data file in the multiple data file case. A file-level dictionary which points to a data file can not be deleted. All frames owned by the deleted file are returned to the available space pool. The BREAK KEY is inhibited during the DELETE process.

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

DELETE-FILE filename

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

DELETE-FILE DICT filename

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

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

DELETE-FILE DATA filename{,dataname}

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 "dataname" is necessary to delete a file from a dictionary with multiple data files.

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

>DELETE-FILE INVENTORY [CR]

Deletes the INVENTORY dictionary, and all associated data files.

>DELETE-FILE DICT TEST/FILE [CR]

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 is not valid.

>DELETE-FILE DATA DEPT,ACCOUNTING [CR]

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

Examples of DELETE-FILE usage.

2.19 COPYING DATA: THE COPY PROCESSOR

COPYING DATA: THE COPY PROCESSOR

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. The general form of the COPY command is:

COPY {DICT} filename item-list {(options)}

The "filename" parameter specifies the source file. The "item-list" consists of one or more item-ids separated by blanks, or an asterisk (*) specifying all items; the "item-list" specifies the items to be copied. 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 differently depending on whether the copy is to the terminal or line-printer, or to a file. This is specified by the presence of the "T" option (copy to terminal), or the "P" option (copy to line-printer). If neither of these options is specified, the copy is to a file.

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

```
TO:
```

The response to this request is in general of the form:

```
{( {DICT} filename)} {item-list}
```

Where:

1) If the data are to be copied to a DIFFERENT FILE, the destination filename is entered ENCLOSED IN PARENTHESES; the word DICT may optionally precede the filename if the data are being copied to a destination dictionary file instead of a data file.

2) If the data are being copied to the SAME file, the parenthetical specification 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 a null is entered to the "TO" request, a copy to the terminal is performed (just as if the original COPY statement had the "T" option).

This is discussed further in the next sections.

2.20 COPYING DATA: FILE TO FILE COPY

This section discusses further the copying of data from one file to another, or within the same file.

In using the COPY operation, multiple items may be specified as the source and as the destination. Multiple item-ids are separated by blanks, unless the item-id itself has embedded blanks, in which case the entire item-id may 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 have been preceded by a SELECT, SSELECT, QSELECT or GET-LIST statement. See the appropriate sections of other chapters for a discussion of these verbs.

When copying from one dictionary to another, the COPY processor does not copy dictionary items which have D/CODE of "D" (that is, the D-pointers). D-pointers must only be created by the CREATE-FILE processor. To recreate both the dictionary and the data sections of on file in a new file, a command sequence such as the example shown below must be used.

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

Copying Items to the Same 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
```

Copying Items to a Different File.

```
>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
```

Recreation of Entire Dictionary and Data Sections.

2.21 COPYING DATA: THE COPY PROCESSOR OPTIONS

This section describes the options that may be specified in the COPY statement. It also describes the method of copying data to the terminal or the line-printer.

FORMAT:

COPY {DICT} filename 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. The table below describes the options used by the COPY processor. Note that some options operate differently depending on whether the copy is to the terminal/line-printer, or is a file copy.

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
```

OPTION	NOTE	DESCRIPTION
D	1	Delete item; the original (source item) is deleted from the file after it is copied.
F	2	Form-feed; each item will cause a new page to begin.
I	1	Item-id list suppress; will inhibit the listing of item-ids.
N	1	New item inhibit; will not copy the items to the destination file unless the item ALREADY EXISTS there. That is, NEW items will not be created if this option is set.
	2	Will inhibit the automatic end-of-page wait.
O	1	Overwrite items option; will copy the item to the destination file EVEN if it already exists on file.
P		Printer copy; copies the data to the line-printer.
S	1	Suppress error messages; messages indicating that items were not copied (messages 409, 415 and 418) will not be printed.
	2	Suppress line-numbers; the line-numbers will not be displayed.
T		Terminal copy; copies the data to the terminal.
X	1	Hexadecimal format; the data is displayed in the hexadecimal form.

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

COPY Processor Options.

SECTION 3

THE ICON/PICK TERMINAL CONTROL LANGUAGE (TCL)

Chapter 3

TERMINAL CONTROL LANGUAGE

THE PICK SYSTEM

USER MANUAL

3.1 INTRODUCTION TO TCL

TCL, meaning Terminal Control Language, is the primary interface between the user and the system. It is from TCL that all other processors (EDITOR, ACCESS, PICK/BASIC, PROC, ASSEMBLY etc.) are invoked. The TCL processor is automatically entered at LOGON and whenever a particular process (such as a LIST or COMPILE) is complete. TCL prompts with 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 perform specified functions or invoke 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 PICK/BASIC Run-Time processor which 'runs' the specified PICK/BASIC program.

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

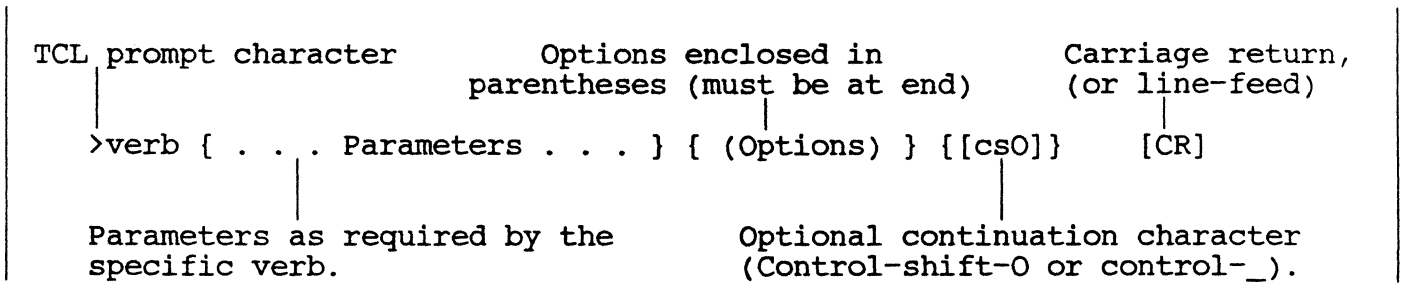
The TCL verbs belong to three major categories. TYPE-I verbs are those which perform specified functions but which do not access data in files. The TIME verb mentioned above is a TYPE-I verb. TYPE-II verbs are those whose functions involve the accessing of data in files. The RUN verb mentioned above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

The user may create any number of synonyms for the verb definition items (and may even remove the pre-defined 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 alphabets, 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 an 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.

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.



General form of a TCL input statement.

CHARACTER	EDITING FUNCTION	COMMENTS
carriage-return (or line-feed)	End of line.	System will take action on TCL statement.
Control-H	Backspace over last Character.	No action if at left margin;. Character echoed by system may be set by the TERM command.
Control-W	Backspace over last word	As above.
Control-X	Delete last line.	No action if at left margin; new line will be started otherwise.
Control-R	Retype last line.	

(Note: the above are system-wide editing functions, and are applicable whenever the system requests data input from the user's terminal.)

Line-editing characters.

3.2 TCL VERB TYPES

There are three basic types of TCL verbs. Type I does not reference a file; type II and ACCESS verbs always reference a file.

TYPE I VERB

The type I TCL verb does not reference a file in the TCL statement. For example, the verb used to attach the magnetic tape unit is:

```
>T-ATT
```

TYPE II VERB

The type II verbs always reference a single file. Typically, one or more explicitly named items (records) in the file may be accessed. Alternately, all items in the file may be accessed. For, example, the verb "ED" invokes the text editor. The command:

```
>ED INVENTORY 1234
```

will access the item "1234" in the INVENTORY file.

ACCESS VERBS

ACCESS verbs have the most generalized syntax. In general, ACCESS verbs specify a single file name, and have a set of selection criteria which is specified to select a subset of the items in the file. Depending on the particular ACCESS verb, further syntactical elements may be present. For example, the statement below is used to list all employees who were born before 1/1/35:

```
>LIST EMPLOYEES WITH BIRTHDATE BEFORE "1/1/35"
```

3.3 TCL-I VERBS TCL-I VERBS

TCL-I verbs do not access a file. The format of the TCL statement is unique to the specific verb, that is, 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. Some TCL-I verbs additionally allow for various parameter specifications.

VERB	DESCRIPTION .....
BLOCK-PRINT	Sends block characters to spooler
CHARGES	Prints current computer usage
CHARGE-TO	Keeps track of computer usage
CLEAR-FILE	Removes all file items from a file or dictionary.
CREATE-FILE	Creates a new file
DELETE-FILE	Deletes an entire file
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.
TABS	Sets tabs for input or output.
TIME	Displays the current time and date.
SP-ASSIGN	Sets up assignment status for the spooler.
SP-STATUS	Spooler and line printer status.
T-ATT	Attaches magnetic tape unit.
T-DET	Detaches the magnetic tape unit.
TERM	Sets or displays terminal characteristics.
TIME	Prints time and date.
WHAT	Displays current system parameters.
WHO	Prints the line number and account name to which any terminal is logged on.

EXAMPLES OF TCL-I VERBS

3.4 TCL-II VERBS TCL-II VERBS

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 Reference Manual). The advantage gained by this restricted format is an enhancement in processing speed since statement parsing is quicker.

FORMAT:

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

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, alternately, all items may be specified via use of the asterisk (*) character). 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-id's, separated by one or more blanks. If an item-id contains embedded blanks or parentheses, it must be surrounded by 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. The specified options are passed to the appropriate TCL-II processor.

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 chapter.

VERB	DESCRIPTION
COMPILE	Compiles a DATA/BASIC program.
CATALOG	Catalogs a DATA/BASIC program.
COPY	Copies data files and dictionaries.
EDIT	Evokes the EDITOR processor.
RUN	Executes a DATA/BASIC program.
RUNOFF	Evokes the word-processer.

Examples of some TCL-II Verbs.

3.5 LOGON AND LOGOFF PROCESSORS LOGON AND LOGOFF PROCESSORS

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.

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

LOGON PLEASE:

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 name (identification) established for him in the system, followed by a carriage-return. If a password has also been established, he may follow his identification with a comma, and then the password, followed by a carriage-return. If the password is not entered as a response to the LOGON PLEASE message, the system will display the message:

PASSWORD:

The system validates the user's identification against the entries in the SYSTEM Dictionary; if it is illegal, the following message is returned:

```
USER-ID?

LOGON PLEASE:
```

The user must then re-enter his identification and password. If the user's identification is valid, but the password is not acceptable, the following message is displayed:

```
PASSWORD?

LOGON PLEASE:
```

The user must then re-enter his identification and password. If the user has successfully logged on to the system) i.e., both the identification and the password have been accepted, the following message is displayed:

```
< WELCOME TO THE PICK SYSTEM >
< time          release                date >


>                                      <------ TCL prompt.
```

where "time" is the current time, "date" is the current date, and "release" is the current PICK Systems release level. The ">" is the TCL prompt character, which indicates that the user may now enter any valid TCL level command.

LOGGING OFF

FORMAT:

```
>OFF
```

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 charge 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.; CHARGE UNITS = m, LPTR PAGES= x >
<       LOGGED OFF AT time      ON  date                  >
```

where "n" is the number of minutes of connect time, "m" is the number of charge units, "time" is the current time, and "date" is the current date, and "x" is the number of line-printer pages generated. The charge-units represent usage of the CPU; it is in tenths of a CPU second.

3.6 LOGTO

The LOGTO verb allows the user to log to another account faster than by going through the OFF and LOGON process.

FORMAT:

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

where "account-name" is that of the new account that the user wishes to logon to, and "password" is the password associated with that account-name. If "password" is not entered, and the account has a password defined, the message:

```
PASSWORD:
```

will be displayed, and the password may then be entered.

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.

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.;  CHARGE UNITS = m, LPTR PAGES= x >>>
```

will be displayed.

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

```
 LOGON PLEASE: SMITH,XYZ [CR]

< WELCOME TO THE PICK OPERATING SYSTEM >
< 09:15:33     RELEASE n    4 JUL 1984 >

>WHO [CR]
7 SMITH

>LOGTO JONES [CR]
PASSWORD: ABC [CR]

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

>WHO [CR]
7 JONES
```

Sample usage of LOGTO verb.

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.

FORMAT:

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. The CHARGE-TO statement will also cause the following message to be displayed:

<<< CONNECT TIME = n MINS.; CHARGE UNITS = m, LPTR PAGES= x >>>

FORMAT:

CHARGES

This will display the logon statistics with the following message:

<<< CONNECT TIME = n MINS.; CHARGE UNITS = m, LPTR PAGES= x >>>

```
LOGON PLEASE: SMITH,XYZ [CR]

   < WELCOME TO THE PICK SYSTEM >
   < 08:15:25      RELEASE n   5 MAY 1984 >

>WHO [CR]
7 SMITH

>CHARGE-TO A001 [CR]
 <<< CONNECT TIME = 0 MINS.; CHARGE UNITS = 7, LPTR PAGES= 0 >>>

>WHO [CR]
7 SMITH*A001

>CHARGES [CR]
 <<< CONNECT TIME = 0 MINS.; CHARGE UNITS = 8, LPTR PAGES= 0 >>>

>CHARGE-TO [CR]
 <<< CONNECT TIME = 0 MINS.; CHARGE UNITS = 9, LPTR PAGES= 0 >>>

>WHO [CR]
7 SMITH
```

Sample usage of CHARGE-TO and CHARGES.

3.8 LOGON PROCS

Upon logon, the Pick Computer System allows for the execution of a PROC with an item-id identical to the user's identification.

When the user has logged on to his account, PICK permits the automatic execution of PROC 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. (See PROC.)

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 listed in the first example (which correspond to an 8 1/2" by 11" page size). These conditions can subsequently be displayed and altered by the TCL verb TERM. As an example, assume that the PROC listed in the second example (which includes a TERM operation) is stored as item SMITH in the user's Master Dictionary (MD). If the user's identification is the word SMITH, then the SMITH PROC will be executed automatically every time the user logs on (i.e., the user's particular terminal characteristics will automatically be set).

	TERMINAL	PRINTER
Page Width:	79 characters	132
Page Body:	24 lines	60
Line Skip:	0	
Line-Feed Delay:	0	
Form-Feed Delay:	0	
Backspace Echo:	8	
Terminal Type:	T	

Initial Terminal Characteristics Automatically Set at Logon Time.

```
Item 'SMITH -- a sample logon PROC.' in MD of user SMITH
001  PQ
002  HTERM 118,44,7,6
003  P
004  X**  TERMINAL CHARACTERISTICS SET **

LOGON PLEASE: SMITH,XYZ  [CR] <-------------------- Logon sequence.

   < WELCOME TO THE PICK SYSTEM >
   < 15:09:50    RELEASE n   13 JULY 1984 >

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

Automatic Execution of Sample PROC.

Terminal and/or line printer characteristics may be displayed or set by a process via the TERM command.

FORMAT:

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

ARGUMENTS:

a is the terminal line length (i.e.,number of characters per line). The a parameter must be in the following range: 16<a<140.
b is the number of print lines per page on the terminal.
c is the number of blank lines per page on the terminal (sum of b and c equals page length).
d is the number of delay or idle characters following each carriage return or line feed. This is used for terminals that require a pause after a carriage return or line feed (i.e., since the CPU generates characters faster than the terminal can accept them).
e is the number of delay characters following each top-of-form.
If e is zero, no form-feed character will be sent to either the terminal or the printer.
If e is non-zero, a form-feed character is also output before each page; if e is ONE, this character is sent to the line-printer, but not to the terminal.
If e is greater than 1, the form-feed character is also sent to the terminal at the beginning of each page, AND that many delay or idle characters is also sent to allow the terminal time to settle after the form-feed.
The form-feed character sent to the printer is always a hexadecimal '0C' (ASCII FF character).
f is the backspace character. An ASCII backspace (control-H) is always input to backspace over (or erase the last character that was input; however, the user may set the actual character echoed to his terminal. This accommodates terminals that cannot physically backspace, or that have a backspace character other than the ASCII backspace. The f parameter should be 21 for the ADDS REGENT terminal, and 8 for the TEC 2402 terminal.
g is the line printer line length.
h is the line printer page length.
t is the terminal type code; this changes the form-feed character sent by the system to match the terminal requirements, and, more importantly, sets the appropriate cursor addressing for the BASIC cursor functions. A few TERMTYPES are:

A - ADDS 580
D - DIALOG
L - LEAR-SIEGLER ADM-3A
d - ICON DT1200
R - ADDS REGENT
T - TV950
V - ADDS VIEWPOINT
X - NO CURSOR ADDRESSING FUNCTIONS

Individual parameters may be null (i.e., as 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.

```
>TERM [CR]

                    TERMINAL PRINTER
          PAGE WIDTH:     79      132
          PAGE DEPTH:     24       64
          LINE SKIP :      0
          LF DELAY  :      1
          FF DELAY  :      1
          BACKSPACE :     21
          TERM TYPE :      R

Standard terminal characteristics set for the ADDS REGENT terminal.

>TERM ,,,,2 [CR]

Resets the FF delay to 2, in order to get a clear-screen on the terminal.

>TERM [CR]

                    TERMINAL PRINTER
          PAGE WIDTH:     79      132
          PAGE DEPTH:     24       64
          LINE SKIP :      0
          LF DELAY  :      1
          FF DELAY  :      2
          BACKSPACE :     21

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

Resets the line-printer page size to 120x48.

>TERM [CR]

                    TERMINAL PRINTER
          PAGE WIDTH:     79      120
          PAGE DEPTH:     24       48
          LINE SKIP :      0
          LF DELAY  :      1
          FF DELAY  :      2
          BACKSPACE :     21
```

Sample usage of the TERM statement.

Tab stops may be set with the TABS statement.

FORMAT:

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

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

where the tabs may be set for input or output, depending on the parameter "O" or "I" following the TABS verb. n1, n2, etc. are up to fifteen tab-stop positions; they must be in ascending numerical sequence.

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 those printing terminals that have a physical tabbing capability.. Do not set output tabs for a CRT! 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 setup 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.

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

>TABS O 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
         O         O         O         O         O         O

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

Examples of TABS statements.

3.11 TIME

The TIME statement displays the current system time and date.

FORMAT:

TIME

TIME is a simple TCL-I verb which returns the current system time and date in external format.

EXAMPLE:

```
>TIME [CR]
09:21:23  11 MAY 1984
```

Example of TIME Verb.

3.12 SLEEP

The SLEEP verb is a TCL-I verb that 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.

FORMAT:

SLEEP x

The "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.

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.

Sample usage of the SLEEP Verb.

3.13 WHO

The WHO statement is a TCL-I verb which is used to display the account-name that a terminal is currently logged on to.

FORMAT:

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".

You may specify a range of lines as well. Any non-numeric character will cause WHO to display all lines and their logon name.

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

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

 >WHO 1-3
01 JOHN
02 SYSPROG
03 UNKNOWN

 >WHO 'SYSPROG'         (displays all lines logged onto the SYSPROG account.)
```

Sample usage of the WHO Verb.

The MSG or MESSAGE statement allows one user to send a message to another user.

FORMAT:

MSG account-name ..... Message text .....

MSG !port-number ..... Message text .....

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 (see SYSTEM MAINTENANCE) 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.

The MSG verb will also allow you 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, by using an asterisk.

EXAMPLE:

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

>MSG JONES HELLO THERE!"%%%%'''%    [CR]
  USER NOT LOGGED ON                          (JONES is not logged on).

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

>MSG !7 HELLO     [CR]                      (Send "HELLO" to line 7)

>MSG !* LOG OFF PLZ    [CR]                 (MSG to all connected terminals)

>MSG !* AUTOMATIC DISK REFORMAT STARTING IN 10 SECONDS.     [CR]
```

Sample usage of the MSG Verb.

Processing can be interrupted by depressing 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 depressed, 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 (refer to the DEBUG documentation in the PICK Assembly Language Manual). The DEBUG prompt character (!) is displayed to prompt the user for a DEBUG command. For users with system privilege levels zero or one, the commands listed in the example are the only DEBUG commands allowed. Users with system privilege level two should refer to the PICK Assembly Language Reference Manual for further DEBUG facilities.

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 zero or one, he must type END or OFF to exit from the DEBUG state. The hardware abnormal conditions are described in the DEBUG section of the Pick Assembly Language manual.

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 LINE FEED	GO. This command causes resumption of process execution from the point of interruption. LINE FEED cannot be used if a process ABORT condition caused the entry to DEBUG.
END	Terminates current process and causes an immediate return to TCL.
OFF	Terminates current process and causes the user to be logged off the system.

DEBUG Commands for Users With System Privilege Levels 0 or 1

3.16 BLOCK-PRINT

The BLOCK-PRINT command will print characters in a 9-by-n block-form on the line printer or the user's terminal, respectively. Any ASCII characters may be printed.

FORMAT:

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. For example, to BLOCK-PRINT JUDY'S JOB, only enclose JUDY'S with double quotes: "JUDY'S" JOB

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

Character-strings to be blocked cannot have more than nine characters. For the BLOCK-PRINT command, 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 520 through 525 may be displayed (refer to the list of error messages in the appendix of this manual).

The BLOCK-PRINT commands use a file named BLOCK-CONVERT to create the blocked characters. A BLOCK-CONVERT file already exists which contains the conversion specifications for all printable ASCII characters (no lower case alphas, however). With this file, characters will be printed as 9-by-12 to 9-by-20 blocks.

If it is desired to change the way any character is printed, it is necessary to change the corresponding item in the BLOCK-CONVERT file. The item-id of the item is the character to be converted. Each item in the file must consist of exactly ten attributes. The first must specify in decimal the number of horizontal bytes in the blocked character to be output (i.e., "n" of the 9-by-n block mentioned above). The second and subsequent attributes provide the conversion specification. These attributes contain one or more values; each value is separated from the preceding by a value mark (ASCII 253). The first character of the first value in each attribute must be "C" or "B"; these signal that the output matrix line of the blocked character begins with a character or a blank, respectively. Immediately following must be the number of characters or blanks (in decimal). The presence of a value mark indicates a switch from character to blank status (or vice versa) and must be followed by the number of bytes to be output. The process continues until the attribute mark at the end of the current line.

EXAMPLE:

```
      BLOCK-PRINT HELLO (CR)        will look like this:


          HH    HH  EEEEEE  LL      LL       OOOOOO
          HH    HH  EE      LL      LL      OO    OO
          HH    HH  EE      LL      LL      OO    OO
          HHHHHHH   EEEEE   LL      LL      OO    OO
          HH    HH  EE      LL      LL      OO    OO
          HH    HH  EE      LL      LL      OO    OO
          HH    HH  EEEEEE  LLLLLL  LLLLLL   OOOOOO


    >BLOCK-PRINT "JUDY'S" JOB (CR)        will look like this:

          JJ  UU    UU  DDDDDD   YY      YY   '''    SSSSS
          JJ  UU    UU  DD   DD   YY    YY    '''   SS   SS
          JJ  UU    UU  DD   DD    YY  YY     '''   SS
          JJ  UU    UU  DD   DD      YY              SSSSS
          JJ  UU    UU  DD   DD      YY                  SS
     JJ  JJ   UU    UU  DD   DD      YY             SS   SS
      JJJJ     UUUUUU   DDDDDD      YY              SSSSS

                        JJ   OOOOOO   BBBBBB
                        JJ  OO    OO  BB   BB
                        JJ  OO    OO  BB   BB
                        JJ  OO    OO  BBBBBB
                        JJ  OO    OO  BB   BB
                    JJ  JJ  OO    OO  BB   BB
                     JJJJ    OOOOOO   BBBBBB
```

Sample usage of the BLOCK-PRINT verb.

3.17 UTILITY PROCS : CT, LISTACC, LISTCONN, LISTDICTS, LISTFILES, LISTPROCS, LISTU, LISTVERBS.

This topic describes various utility PROC's.

CT

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

LISTACC {account-name}...

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

LISTCONN

LISTCONN file-name {LPTR}

This PROC sorts all connectives in any dictionary and lists them on the terminal (or Line-PrinTeR if specified).

LISTDICTS

LISTDICTS file-name {LPTR}

The LISTDICTS PROC sorts all attribute synonym definition items in any dictionary and lists them on the terminal (or Line-PrinTeR if specified).

LISTFILES

LISTFILES file-name {LPTR}

The LISTFILES PROC sorts all file or file synonym definition items in any dictionary and lists them on the terminal (or Line-PrinTeR if specified).

LISTPROCS

LISTPROCS file-name {LPTR}

The LISTPROCS PROC sorts all PROC's in any file or dictionary and lists them along with a brief abstract on the terminal (or Line-PrinTeR if specified).

LISTU

LISTU

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

LISTVERBS

LISTVERBS file-name {LPTR}

The LISTVERBS PROC sorts all verbs (not PROC's in any dictionary and lists them on the terminal (or Line-PrinTeR if specified).

3.18 VERB DEFINITION ITEMS IN M/DICT

Each TCL-I, TCL-II, or ACCESS verb is defined as an item in the user's Master Dictionary (MD).

Each verb definition resides as an item in the user's Master Dictionary. The item-id (i.e., attribute zero) of a verb definition item is the verb name itself. The user may create any number of synonyms for the verb definition items (and may even remove the pre-defined 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.

ATTRIBUTE NUMBER	DESCRIPTION
0	This is the item-id, which is the name of the verb.
1	Must contain: Pc 1 P identifies the MD item as a verb definition item. The single character c is passed to the defined processor. If c is Q, the item is a PROC not a verb.
2	This attribute defines the processor entry point to which TCL passes control (i.e., the mode-id in hex). See PICK ASSEMBLER Manual.
3	Secondary transfer point. Use depends on attributes 1 and 2.
4	Tertiary transfer point. Use depends on attributes 1 and 2.
5	TCL-II parameter string. These parameters govern treatment of the items retrieval by TCL-II verbs to be passed to the processor whose entry point is defined in attribute three. 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 required. U - Items will be updated by processor. Z - Final entry required on EOI.

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

Verb Definition Item in MD.

SECTION 4

THE ICON/PICK EDITOR

Chapter 4

EDITOR

THE PICK SYSTEM

USER MANUAL

4.1 EDITOR PROCESSOR : AN INTRODUCTION

The EDITOR is a processor which permits on-line interactive modification of any item in the data base. The EDITOR may be used to create and/or modify PICK/BASIC programs, PROC's, assembly source, data files, and file dictionaries. The EDITOR uses the current line concept; that is, at any given time there is a current line that can be listed, altered, deleted, etc. The EDITOR includes the following features:

- Two variable length temporary buffers
- Absolute and relative current line positioning
- Line number prompting on input
- Merging of lines from the same or other items
- Character string location and replacement
- Conditional and unconditional line deletion
- Input/Output formatting
- Prestoring of commands

EDITOR COMMAND AND EXAMPLE CONVENTIONS

CONVENTION	MEANING
UPPER CASE	Characters printed in upper case are required and must appear exactly as shown.
Lower case	Characters or words printed in lower case are parameters to be supplied by the user (i.e., line number, data, etc.).
{}	Braces surrounding a parameter indicate that the parameter is optional and may be included or omitted at the user's option.
"string"	A "string" is a sequence of characters delimited by any non-numeric character (except a blank or a minus sign) that does not appear within the body of the "string" itself. (A further description of "string" is presented in the topic describing the Editor syntax).

Conventions Used in EDITOR command Formats

CONVENTION	MEANING
* TEXT	An asterisk preceding text represents the user's input.
TEXT	Capitalized text represents output printed by the EDITOR
[CR]	This symbol represents a carriage return.

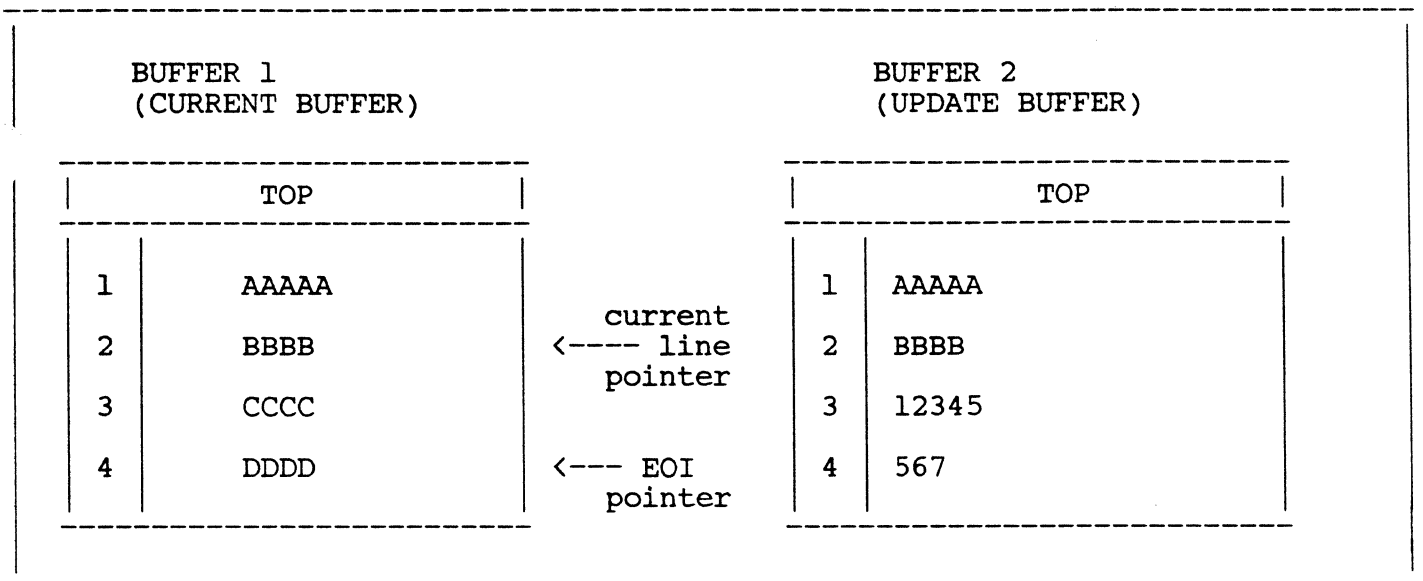
Conventions Used in EDITOR Examples

The EDITOR uses two data areas (buffers) to edit an item. The item is copied into one buffer and updates are assembled in the other. An F command merges the updates with the item and then toggles the function of the buffers.

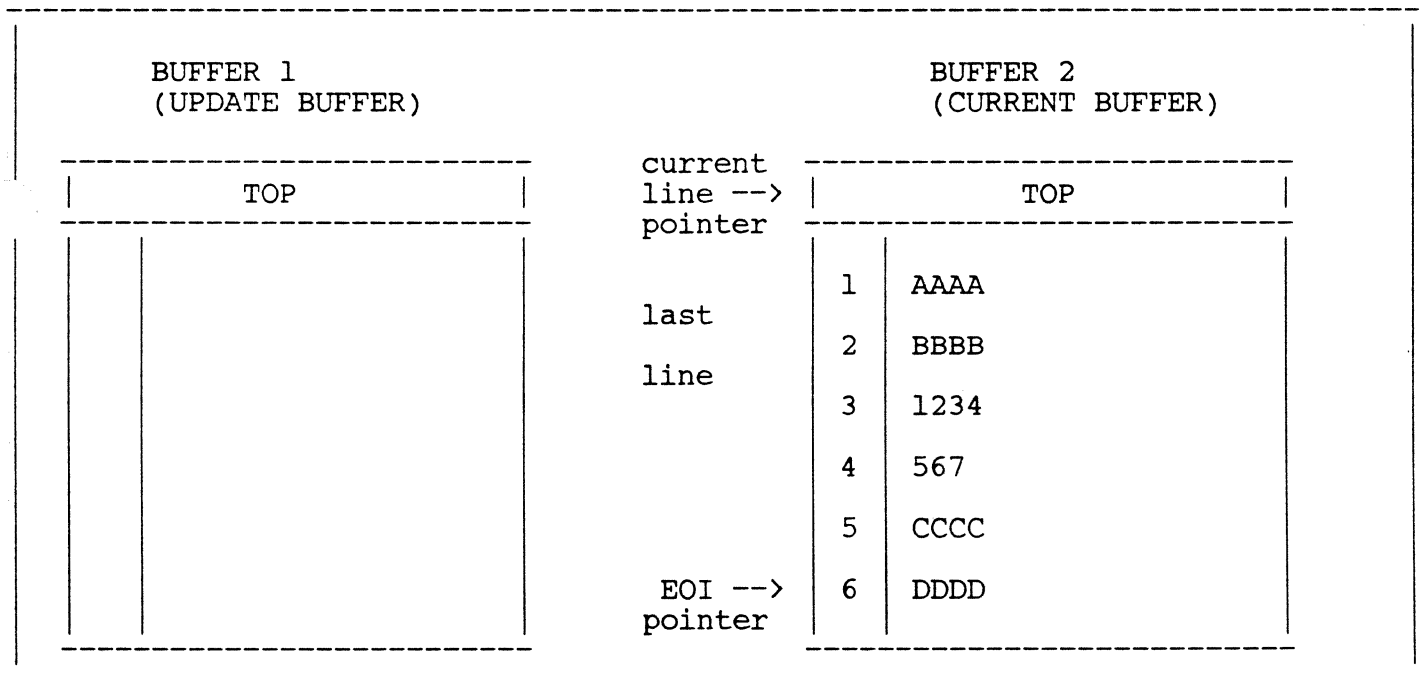
The EDITOR uses two variable length temporary buffers (Buffer 1 and Buffer 2) to create or update an item. When the EDITOR is entered, the item to be edited is copied from the file to Buffer 1 (the Current Buffer). Each line (attribute) of the item is associated with a line number. A current line pointer points to the current line of the item, and an EOI (End-of-Item) pointer points to the last line of the item. EDITOR operations are performed on one line at a time (the current line) in an ascending line number sequence from TOP (line 0) to EOI. As an EDITOR operation is performed on a line, the modified line and all previous lines are copied to Buffer 2 (the Update Buffer).

The editing process continues working on Buffer 1. As lines in the item are changed (or lines are inserted or deleted), the EDITOR builds a new updated version of the item in Buffer 2. Updating must thus continue in an ascending line number sequence until a F command is entered. The F command merges the updates with the previously existing item, and an automatic resequencing of the item takes place. The F command does not permanently file an item; it completes the copy to the Update Buffer causing all lines to be resequenced and the EOI pointer to be repositioned. It then switches (toggles) the function of the buffers, so that Buffer 1 becomes the Update Buffer and Buffer 2 becomes the Current Buffer. Editing then occurs in Buffer 2 with new modifications assembled in Buffer 1. This toggling of buffers can go in indefinitely until the item is permanently filed away via a File Item (FI) or File Save (FS) command.

This editing process is exemplified in the following examples. The first example shows a four-line item in Buffer 1 (the Current Buffer) with the current line pointer positioned at line 2. Two lines ("1234" and "567") are then inserted after line 2 as can be seen in Buffer 2 (the Update Buffer). When an F command is issued, the buffers are toggled and the situation is as shown in the second example. Here Buffer 2 has become the Current Buffer. Further modifications made to the item will be assembled in Buffer 1 which has now become the Update Buffer.



Editing Example Before F Command



Editing Example After F Command.

| To enter the EDITOR, the EDIT verb is entered at the TCL level. |

FORMAT:

ED{IT} {DICT} file-name {item-list} {(options)}

The "item-list" parameter consists of one or more item-id's separated by blanks, or an asterisk character (*) specifying all items in the specified file. If multiple item-id's are specified, then the first item specified will be edited first; when the EDITOR then is terminated via a File Item (FI), File Delete (FD), or Exit (EX) command, then the EDITOR will automatically be re-entered and the next item will be edited; and so on.

If the DICT option is used, the specified item(s) in the dictionary section of the specified file will be edited. If DICT is omitted, the specified item(s) in the data section of the specified file will be edited.

If a select-list is in effect (by using a SELECT, SSELECT, QSELECT or GET-LIST), the item-list is omitted; the item-ids are obtained from the select-list in this case.

Editor options are specified as a single character; multiple options may be separated by commas.

EDITOR OPTIONS:

A	Turns on the assembly-code formatting option; see "AS" command.
S	Turns on the suppress-line numbers on suppress object-code option; see "S" command.
M	Turns on the macro expansion flag; see "M" command.
P	Sends all system ouput to the line printer.
Z	Suppresses "TOP" and "EOI" messages.

NOTES ON THE EDITOR:

Once the EDITOR has been entered, the following will be printed:

```
TOP
.
```

The current line pointer is set to line zero, and an EDITOR command is awaited (i.e., the period prompt character (.) indicates that an EDITOR command is to be entered).

If the specified item does not already exist on file, the message "NEW ITEM" will be printed prior to the "TOP" message. Furthermore, if multiple item-id's were specified, then the item-id of item currently being edited will be printed.

As noted in the discussion of file structure, the elements subsidiary to files are items. Structurally they are made up of attributes, and functionally they all are seen by some processor as data; but intuitively one may consider items to be of two types: text or data. A data item is typified by the condition that the meaning of a data string depends upon which attribute it is in. A text item is a sequential string using the attribute mark and count at most to delimit sub-strings. Data strings include Attribute defining items found in dictionaries, and data items in files to be processed by ACCESS, PICK/BASIC or User exits, wherein individual lines are properly referred to as attributes. Text items are made up of lines, which are structurally identical to the attributes of data items, but which do not have meaning by virtue of their attribute location. Text items include PICK/BASIC and Assembler language programs, Procs, and the items processed by RUNOFF.

The EDITOR has the capacity to create, modify, and delete both data and text items anywhere in the System, within the constraints of the user's account's privilege level and update lock codes, without respect to the type of item or its end use.

The EDITOR displays attributes as lines, so that the attribute mark count within the item and the line number displayed by the EDITOR are identical. Note that attribute zero is the item-id.

EXAMPLES:

```
* >ED F1 I1 I2 I3  [CR] <--------- EDIT verb (with multiple item-id's).
  I1 <---------------------------- Item I1 is edited first.
  TOP
* .EX <--------------------------- Exit command (exits EDITOR).
  EXIT
  I2 <---------------------------- EDITOR automatically re-entered
TOP                                to edit next item (I2).
* .EX <--------------------------- Exit command.
  EXIT
  I3 <---------------------------- EDITOR automatically re-entered.
  NEW ITEM <---------------------- Shows that I3 is a new item.
  TOP
* .EX <--------------------------- Exit command.
  EXIT
  > <----------------------------- Returns to TCL level.
```

Sample Usage of the EDIT Verb.

| This section describes the syntax of EDITOR commands. |

EDITOR commands are one or two letter mnemonics which must appear as the first non-blank input character. Command parameters follow the command; blanks may be inserted between parameters for clarity if desired, but embedded blanks in parameters are not permitted.

EDITOR commands can be entered either in upper or lower case. This is especially convenient when editing text items, when the terminal may be in the lower-case mode.

4.4.1 EDITOR "strings"

Certain EDITOR commands use a "string" which may be defined as a series of characters that is surrounded, or delimited by a pair of identical, non-numeric characters that do not appear within the string itself. Lower case alphabetic characters are not valid as delimiters.

VALID STRINGS

```
/123 MAIN ST./                  /replacement of the/
.abc 123 DEF.                   " $ 9876.54 "
;Open Architecture: ;           . "PICK/BASIC is .
AThis test stringA              Z That test string Z
```

For convenience, the closing delimiter of the "string" is necessary only if further parameters follow the string specification, or trailing blanks are to be included as part of the "string".

4.4.2 COLON : EDITOR DELIMITER

The "string" is used in EDITOR commands that specify a search for matching data in the item. The COLON (:) is a reserved delimiter; if used, it indicates that a column-dependent correspondence between characters in the string and characters in the line is necessary for a match.

```
:LOOP :
```

would attempt to find the matching characters "LOOP " in columns 1 through 5 of the line; however, the string:

```
/LOOP /
```

would attempt to find the matching characters "LOOP " anywhere within the line.

4.4.3 UP-ARROW : WILDCARD EDITOR CHARACTER

The up-arrow (^) is a reserved character within the body of the "string". The up-arrow is a wildcard character used with L(ocate) and R(eplace) EDITOR commands. It indicates that any character in the corresponding position in the line is acceptable as a match. Note that this feature may be nullified by using the "^" Command. For example, the string:

```
/AB^CD/
```

would attempt to find the matching characters "AB", then any character whatsoever, then "CD" in the line. This feature may be deactivated by using the "^ " character alone at the command prompt. Entering it again will toggle the feature back to activated. The EDITOR outputs /ON\ or /OFF\ accordingly.

COMMAND NAME	COMMAND FORMAT
Again	A
Assembler Format ON/OFF	AS
Bottom	B
Column Number List	C
Current Line	?
Delete	DE{n}
Delete	DE{n}"string"{p{-q}}
Exit	EX{K}
File Delete	FD
File Item	FI
File Save	FS
	F
Goto	Gn
Goto	n
Input	I
Insert	I data
List	L{n}
Locate	L{n}"string"{p{-q}}
Macro expansion	M
Merge	ME{n}"item"{m}
Next	N{n}
Prestore	P command
Prestore Call	P
Replace	R
Replace	RU{n}"string 1"string 2" {p{-q}}
Suppress ON/OFF	S
Tab	TB xx xx xx ... xx
Top	T
Up	U
X	X
XF	XF
Zone	Z{p{-q}}

EDITOR Command Summary.

The commands that are provided for controlling the the current line pointer and for listing the item being edited, are described in this section.

4.5.1 "L" - LIST COMMAND : EDITOR

FORMAT:

L{n}

This command causes n lines to be listed, starting from the current line plus one. If n is omitted, only one line is listed. If n is greater than or equal to the number of lines from the current line to the EOI, then all the lines down to the EOI will be listed.

If a List command is issued when the current line pointer is at the EOI, then the next n lines starting from line 1 will be listed. The List command positions the current line pointer at the last line listed.

4.5.2 NULL COMMAND <CR> : EDITOR

FORMAT:

<CR>

The Null command is executed by entering a carriage return only. This command is identical to a List command where n is omitted. The next line is listed and the current line pointer is advanced one line. This command is included for convenience when stepping through lines in an item.

4.5.3 "U" - UP COMMAND : EDITOR

FORMAT:

U{n}

The Up command decrements the current line pointer by n lines, and then lists the new current line. If n is omitted or is zero, the current line will be listed.

4.5.4 "N" - NEXT COMMAND : EDITOR

FORMAT:

N{n}

This command increments the current line pointer by n lines, and (one line if n is omitted), and then lists the new current line.

For all of the above commands, the message TOP will be printed if the current line pointer is set to zero, and the message EOI m (where m is the last line number of the item) will be printed if the pointer is set to the EOI.

4.5.5 "G" GOTO COMMAND : EDITOR

FORMAT:

Gn OR n

These commands position the current line pointer and list line n.

4.5.6 "T" TOP COMMAND : EDITOR

FORMAT:

T

TOP sets the current line pointer to zero.

4.5.7 "B" BOTTOM COMMAND : EDITOR

FORMAT:

B

Bottom sets the current line pointer to EOI.

On the above commands, the message TOP will be printed if the current line pointer is set to zero, and the message EOI m

```
* >EDIT FILE1 ITEM  [CR]
  TOP
* .P    [CR] <---------------------- P0 - Prestored command (lists 22 lines
  001 AAAAA   ---
  002 BBBBB     |
  003 CCCCC     | <------------------ Item consists of only 6 lines,
  004 DDDDD     |                     so entire item is listed.
  005 EEEEE     |
  006 FFFFF   ---
  EOI 6
* .  [CR] <------------------------- Null command (since current line
  TOP                                pointer is at EOI, the 1st line
  001 AAAAA                          is listed).
* .  [CR] <------------------------- Null command (lists next line).
  002 BBBBB
* .N2  [CR] <----------------------- Next command (goes down 2 lines
  004 DDDDD                          and lists line).
* .U2  [CR] <----------------------- Up command (goes up 2 lines and
  002 BBBBB                          lists line).
* .N9 [CR] <------------------------ Next command; since the item has
  006 FFFF                           only six lines, the last line
  EOI 6                              is listed.
* .L  [CR] <------------------------ List command (since current line
  TOP                                pointer is at EOI, the 1st line
  001 AAAAA                          is listed).
* .L3  [CR] <----------------------- List command (lists next 3 lines).
  002 BBBBB
  003 CCCCC
  004 DDDDD
* .T  [CR] <------------------------ Top command (goes to line 0).
  TOP
* .P    [CR] <---------------------- Prestored command (list 22 lines).
  001 AAAAA   ---
  002 BBBBB     |
  003 CCCCC     | <------------------ Item consists of only 6 lines,
  004 DDDDD     |                     so entire item is listed.
  005 EEEEE     |
  006 FFFFF   ---
  EOI 6
* .G5  [CR] <----------------------- Goto command (lists line 5).
  005 EEEEE
* .B  [CR] <------------------------ Bottom command (goes to EOI).
  EOI 6
* .L  [CR] <------------------------ List command (since current line
  TOP                                pointer is at EOI, the 1st line
  001 AAAAA                          is listed).
* .3  [CR] <------------------------ Goto command (lists line 3).
  003 CCCCC
* .T  [CR] <------------------------ Top command (goes to line 0).
  TOP
  .
```

Sample Usage of Line Control Commands.

4.6 STRING MATCH LOCATING : EDITOR

The Locate command causes a search for characters that match a specified string. The Again command repeats the last Locate command issued.

4.6.1 "L" - LOCATE COMMAND : EDITOR

FORMAT:

L{n}"string"{p{-q}}

This command causes a search for characters matching the "string".

The search is restricted to column p, or columns p through q, if specified. If q<p, q=p is assumed. If the delimiter used in the Locate command is a colon, ":", then only matching strings starting in the first column specified (= p) will be located.

If n is not specified, the next ocurrence of "string" is located, and that line is listed; the current line pointer is set at the line that is listed. If n is specified, n lines, starting from the current line plus one, are scanned for the occurrence of "string"; all lines in which the "string" is found are listed. The current line pointer is incremented by n, and therefore might not be located at the last line listed.

The scan always begins from the current line plus one.

4.6.2 "A" - AGAIN COMMAND : EDITOR

FORMAT:

A

The Again command repeats the last Locate command issued.

```
* >ED F1 ABC  [CR]
  TOP
* .P    [CR]
  001 ABCDEFG   --
  002 12ABCDEFG   |
  003 BCDEFG      | <-------- This is what item ABC looks like.
  004 ABC         |
  005 ABCDEFG   --
  EOI 5
* .T  [CR]
  TOP
* .L"ABC  [CR] <-------------- Locate command (locates next line with
                               "ABC").
  001 ABCDEFG <--------------- Line 1 located.
* .T  [CR]
  TOP
* .L5/ABC/  [CR] <------------ Locate command (scans 5 lines and
  001 ABCDEFG    --            locates lines containing "ABC").
  002 12ABCDEFG    |
  004 ABC          | <--------- Lines 1, 2, 4, and 5 located.
  005 ABCDEFG    --
  EOI 5
* .T  [CR]
  TOP
* .L5<A<3-4  [CR] <----------- Locate command (locates "A" in columns
                               3 thru 4).
  002 12ABCDEFG <------------- Line 2 located.
  EOI 5
* L5:ABCD:  [CR] <------------ Locate command (locates "ABCD" column
  TOP                          dependent; i.e., must be in columns 1
                               thru 4).
  001 ABCDEFG  --|
  005 ABCDEFG  --| <---------- Lines 1 and 5 located.
  EOI 5
* .L5:^^AB:  [CR] <----------- Locate command (locates "AB" in columns
  TOP                          3 thru 4).
  002 12ABCDEFG <------------- Line 2 located.
  EOI 5
* .L:^B:  [CR] <-------------- Locate command (locates next line with
  TOP                          "B" in column 2).
  001 ABCDEFG <--------------- Line 1 located.
* .A  [CR] <------------------ Again command (repeats last Locate).
  004 ABC <------------------- Line 4 located.
* .A  [CR] <------------------ Again command (repeats last Locate).
  005 ABCDEFG <--------------- Line 5 located.
```

Sample Usage of Locate and Again Commands.

The Input command is used for data entry. The user may create a new item, or may insert or add lines to an already existing item.

4.7.1 "I" - INPUT COMMAND : EDITOR

"I" - INPUT COMMAND : EDITOR

FORMAT:

I

The Input command, when issued, causes the EDITOR to enter the Input Environment. All subsequent lines input by the user are then considered as data input lines to the item, until the user exits the Input environment.

If the Input command is issued for a new item which has not previously been edited, the new lines will be input to the item starting at line one. The EDITOR will request data lines by prompting with the line number to which data are being entered.

If the Input command is issued for an item already containing data, then the new lines will be inserted following the current line. Input will be prompted with the current line number, after which the lines are being inserted, followed by a plus sign. If the current line pointer is at line zero (TOP), input lines will be inserted before the first line of the item with a prompt of "000+".

A null input (carriage return or line feed only) will cause the EDITOR to exit the Input Environment and await the next EDITOR command. (If a null line is required in the item, it is necessary to create the line with a fill character and then replace the fill character with a null via the Replace command; refer to the topic describing the Replace command. The Insert command can also be used to insert null lines). If there is an error in the current input line, the user can execute a carriage return twice, to enter the line and exit the input environment, then execute a Replace-string operation to fix the error, and then reenter the input environment without executing an F command, except on initial input, as below.

The user should note that when the Input Environment is initially exited for a new item, an automatic F command will be executed by the EDITOR, thus toggling the function of the EDITOR buffers and allowing the newly entered lines to be listed.

If an input line is too long to fit on one physical line, the line continuation character (control-shift 0) may be entered at the end of the physical line and the input line may then be continued on the next physical line. The line-continuation character must be immediately followed by a carriage return or line feed.

EXAMPLES:

```
* >EDIT AFILE AITEM  [CR]
   NEW ITEM <------------------------- Note that this is a new item.
   TOP
  .I  [CR] <--------------------------- Input command.
* 001 INPUT     --| <------------------ Lines being input.
* 002 DATA [CR] --|                       .
* 003  [CR] <-------------------------- Input terminated.
  TOP <------------------------------- Automatic F command has been
                                        executed.
* 003  [CR] <-------------------------- Input terminated.
  TOP <------------------------------- Automatic F command has been
                                        executed.
* .L2 <------------------------------- List command.
  001 INPUT
  002 DATA
  EOI 2
   .
```

Sample Usage of Input Command for New Item.

```
* >EDIT TESTFILE TESTITEM  [CR]
  TOP
* .P    [CR] <--------------------- Prestored command (list 22 lines).
  001 LINE 1   --
  002 LINE 2    | <----------------------- This is what item currents
  003 LINE 3   --                          contains.
  EOI 3
* .T  [CR] <------------------------------ Top command.
  TOP
* .I  [CR] <------------------------------ Input command.
* 000+ NEW LINE A  [CR] <----------------- New line input.
* 000+  [CR] <---------------------------- Input terminated.
* .G2  [CR] <------------------------------ Goto command.
  002 LINE 2
* .I  [CR] <------------------------------ Input command.
* 002+ NEW LINE B  [CR] <----------------- New line input.
* 002+  [CR] <---------------------------- Input terminated.
* .F  [CR] <------------------------------ F command toggles buffers.
  TOP
* .P    [CR] <---------------------------- Prestored command (list 22 lines
  001 NEW LINE A
  002 LINE 1
  003 LINE 2
  004 NEW LINE B
  005 LINE 3
  EOI 5
   .
```

Sample Usage of Input Command for Previously Edited Item.

4.8 INSERTING DATA : EDITOR

The Insert command is used to insert one new line. The Merge command is used to insert one or more lines by merging lines from the same item, or from another item in the same file, or another item in a different file.

4.8.1 "I" - INSERT COMMAND : EDITOR

FORMAT:

I data

The user enters an "I", followed by one blank, followed by the data to be inserted. The specified data will be inserted as a new line after the current line. Note that the data to be inserted must be separated from the "I" by only one blank; all other blanks will be considered as part of the line to be inserted. The line continuation character (control-shift O) cannot be used to continue data beyond one physical line.

The Insert command is most convenient for either inserting only one line of data (rather than using the Input command), or for INSERTING A NULL LINE; the latter is done by entering "I" and one space, followed by a carriage-return. One may also insert a string of Attribute marks to generate a string of null lines. This feature is particuarly useful when entering Dictionary items, which use null lines within their structure.

4.8.2 "ME" - MERGE COMMAND : FROM THE SAME FILE

FORMAT:

ME{n}/item-id/{m}

This command causes n lines (starting from line number m) of the item whose item-id specified by /item-id/ to be merged (inserted) into the item being edited. The lines will be inserted following the current line. The item specified by /item-id/ must be in the same file as the item being edited. A value of one will be assumed for both n and m if either or both are omitted. If /item-id/ is null (//), lines will be merged from the item being edited itself, as it stands in the current buffer, thus duplicating the specified lines in the item.

The user should note that if the item from which lines are to be merged is not on file, the message "NOT ON FILE" will be printed.

4.8.3 MERGE COMMAND : FROM OTHER FILES

The extended syntax requires the use of the delimiters "(" and ")" in place of the "/" delimiter used above. They thus become reserved when using the merge command in the sense that the colon ":" is reserved when using the locate, replace, and delete commands. In this case there is the further peculiarity that "(" and ")" are not the same character, whereas any character may normally be used as a delimiter, so long as all the delimiters in a particular string are identical.

FORMAT:

ME{n}({DICT} [FILENAME] {ITEMNAME}){m}

The use of DICT is conventional. It means the same thing here as is does at TCL in the reference to files, and in the COPY processor. If there is no item-id specified, then the processor defaults to the item-id of the item being edited at the moment. This is useful if one wishes to get a copy of an item into a test file and edit it quickly, or if one wishes to assure that the item will not be filed inadvertently over the old copy. Combined with the prestore command structure and the global replace command, some very powerful things can be done very quickly and easily.

4.8.4 MERGE COMMAND DEFAULTS

There are certain other defaults which apply to the merge command, and which are carried over into this extended form which will be noted below as a reminder.

ME{n}({DICT} [FILENAME] {ITEMNAME}

This form does the same thing as above, except that the starting line number defaults to line 1 in the merge source item.

ME({DICT} [FILENAME] {ITEMNAME}){m}

This does the same thing as above, except that starting-line-number is the only line which is merged into the destination item. As such, the line may then be modified using the replace command, as noted above.

ME({DICT} [FILENAME] {ITEMNAME}

This simply returns the first line of the merge source item. Note that the trailing right parenthesis is optional if the starting line number defaults to the first line of the source.

4.8.5 MINIMAL MERGE

Obviously, these defaults all apply to the normal merge statement, leading to the minimal form 'ME/', which simply inserts the first line of the item currently being edited into the current location in the item, which is useful if you wish to put a given line in several different places in an item.

EXAMPLES:

```
* >EDIT ABC ITEM5  [CR]
  TOP
* .P
  001 ABCDEFG --| <-------------------- This is what ITEM5 looks like.
  002 HIJK    --|
  EOI 2
* .Gl  [CR] <--------------------------- Goto command.
  001 ABCDEFG
* .I 12345  [CR] <---------------------- Insert command.
* .F  [CR] <---------------------------- F command (toggles buffers).
  TOP
* .P    [CR]
  001 ABCDEFG   --
  002 12345       | <------------------- Here is ITEM5 after insertion.
  003 HIJK      --
  EOI 3
  .
```

Sample Usage of Insert Command.

```
* >EDIT FILEl ITEMl  [CR]
  TOP
* .P    [CR]
  001 11111   --
  002 22222     | <------------------- This is what ITEMl looks like.
  003 33333   --
  EOI 3
* .EX  [CR] <-------------------------- Exit command (exits EDITOR).
  EXIT
* >EDIT FILEl ITEM2  [CR]
  TOP
* .P    [CR]
  001 AAAAA   --
  002 BBBBB     | <------------------- This is what ITEM2 looks like.
  003 CCCCC   --
  EOI 3
* .G2  [CR] <-------------------------- Goto command.
  002 BBBBB
* .ME2"ITEMl"l  [CR] <----------------- Merge 2 lines from ITEMl
                                        starting at line 1.
* .F  [CR] <--------------------------- F command (toggles buffers).
  TOP
* .P    [CR]
  001 AAAAA   --
  002 BBBBB     |
  003 11111     | <------------------- Here is ITEM2 after the 2 lines
  004 22222     |                      from ITEMl have been merged.
  005 CCCCC   --
  EOI 5
  .
```

Sample of Merge Command.

| The Delete command causes one or more lines to be deleted from the item. |

4.9.1 "DE" - DELETE COMMAND (SIMPLE) : EDITOR

"DE" - DELETE COMMAND (SIMPL

FORMAT:

DE{n}

This command causes n lines to be deleted (one if n is omitted), starting from the current line. The current line pointer is set to the line after the deletion, allowing further Editor command sequences.

4.9.2 "DE" - DELETE COMMAND (STRING SEARCH) : EDITOR

FORMAT:

DE{n}"string"{p{-q}}

The complex form of the Delete command causes a search for characters matching the specified "string" (see EDITOR command syntax). If n is not specified, n defaults to 1. If n is specified, n lines, starting from the current line, are scanned for the occurrence of "string"; all lines in which the "string" is found are deleted. Lines that are deleted are listed. The current line pointer is set to the line after the span of the Delete command (or n lines).

The search for the specified "string" is column-dependent if the delimiter used in the "string" is a colon, or if parameters p, or p and q are used. If the colon is used, the Editor defaults to column 1 for the "string" match, regardless of any p or q parameters. If p is used by itself, the search starts in column p and continues scanning the remaining line for a match. If both p and q are used, then the scan will match all "strings" whose first character is in column p or greater, while at the same time, the last character of the "string" falling before or at column q. If q<p, q=p is assumed.

The user should note that the scan always begins from the current line.

This is similar to the simple Delete command which starts with the current line and then continues for the next (n-1) lines.

EXAMPLES:

```
* >ED TEST ITEM.1  [CR]
  TOP
* .P    [CR]
  001 123XYZ      --
  002 AAAAAAA       |
  003 XYZ123        | <----------- This is what item ITEM.1 looks like.
  004 ABABABAB      |
  005 12345         |
  006 AA          --
  EOI 6
* .G5  [CR] <------------------- Goto command.
  005 12345
* .DE2  [CR] <----------------- Delete command (deletes 2 lines).
  EOI 6
* .F  [CR] <-------------------- F command (toggles buffers).
  TOP
* .P    [CR]
  001 123XYZ     --
  002 AAAAAAA      | <----------- Here is item ITEM.1 after lines 5
  003 XYZ123       |              and 6 have been deleted.
  004 ABABABAB   --
  EOI 4
* .T  [CR]
  TOP
* .DE99/123   [CR] <------------ Delete command (deletes lines con-
                                 taining "123").
  001 123XYZ  --| <------------- Deleted lines are listed.
  002 XYZ123  --|
  EOI 4
* .F  [CR]
  TOP
* .P    [CR]
  001 AAAAAAA  --| <------------ Here is item ITEM.1 after deletion.
  002 ABABABAB --|
  EOI 2
* .DE:^B    [CR] <-------------- Delete command (deletes lines with "B"
                                 in column 2).
  002 ABABABAB <---------------- Deleted line is listed.
  EOI 2
* .F  [CR]
  TOP
* .P    [CR]
  001 AAAAAAA <----------------- Here is item ITEM.1 after deletion.
  EOI 1
  .
```

Sample Usage of Delete Commands.

The Replace command may be used to replace a number of lines, or may be used to replace one character string with another character string (in one or more lines). The Replace command also allows several executions of the replace on a single line. The "U" option allows replacement of all copies of a string within a line with the specified replacement string.

4.10.1 "R" REPLACE COMMAND (SIMPLE) : EDITOR

FORMAT:

R{n}

The Replace command takes on two general forms. The simple form causes the Input Environment to be entered (see Input command). Input is requested for data to replace n lines (one if n is omitted), starting from the current line. The Input Environment is exited when either:

1) Data for the specified number of lines has been entered, or

2) A null line (i.e., carriage return or line feed only) is entered.

In the latter case, the remainder of the lines (including the line which received the null input) will remain unchanged. The current line pointer points to the next line in the current buffer to be edited.

4.10.2 "R" - REPLACE COMMAND (STRING SEARCH) : EDITOR

FORMAT:

R{U}{n}/string 1/string 2/{p{-q}}

This form of the Replace command causes a search for characters matching "string 1" (see EDITOR command syntax). If n is not specified, then only the current line is scanned for "string 1". If "string 1" is located then it is replaced by "string 2". If n is specified, then n lines which includes the current line are scanned. The first occurrence of "string 1" in each line is replaced by "string 2". Lines that are changed are listed in their updated form.

4.10.3 "RU" - REPLACE COMMAND (UNIVERSAL STRING SEARCH) : EDITOR

This option is indicated by simply using the form RU, as noted by the {U} in the above format.

This form of the Replace command allows the replacement of all cases of "string 1" with "string 2" in the line or lines specified. The option allows multiple-line replacements using the form RUn for the form Rn, and otherwise is identical to the "R" format.

COLUMN SPECIFICATIONS:

As with the "DE" command, if the delimiter is a colon ":", then the column specification defaults to column 1, regardless of any p or q parameters. If only p is used, then the scan begins in column p and continues for the rest of the line until a match is found. If the "RU" form is used, the scan will continue searching for all string matches after column p. If both p and q are present, a match is made if the first character of "string 1" falls in column p or greater while at the same time having the last character of "string 1" fall before or at column q. If q<p, q=p is assumed. Only one delimiter separates "string 1" and "string 2" in the complex form of this command, and the third delimiter may be left out if the column specification is not needed. Any non-numeric character not in "string 1" and "string 2" may be used as the delimiter.

The protocols above are identical for the string locate and the form of the delete which deletes lines which contain a given string.

```
* >ED F1 ABC  [CR]
  TOP
* .P   [CR]
  001 ABCDEF    --
  002 ABCDEF     | <------------ This is what item ABC looks like.
  003 ABCDEF    --
  EOI 3
* .T  [CR]
  TOP
* .R2  [CR] <-------------------- Replace command (replaces 2 lines).
* 001 123ABC  [CR]   --| <------ Replacement lines being input.
* 002 XXXXXAB  [CR]  --|
* .F  [CR] <--------------------- F command (toggles buffers).
  TOP
* .P   [CR]
  001 123ABC   --
  002 XXXXXAB   | <------------- Here is item ABC after replacement.
  003 ABCDEF   --
  EOI 3
* .T  [CR]
  TOP
* .R3"AB"HHH"  [CR] <----------- Replace command (replaces "AB" with
  001 123HHHC   --               "HHH").
  002 XXXXXHHH   | <------------ The 3 lines in which replacement took
  003 HHHCDEF   --               place are listed.
  EOI 3
* .F  [CR]
  TOP
* .R3/HHH/S/1-3  [CR] <--------- Replace command (replaces "HHH" in
                                 columns 1 thru 3 with "S").
  003 SCDEF <------------------- Line in which replacement took place
  EOI 3                          is listed.
* .F  [CR]
  TOP
* .R3/HHH//  [CR] <------------- Replace command (replaces "HHH" with
                                 null).
  001 123C   --| <-------------- Lines in which replacement took place
  002 XXXXX  --|                 listed.
  EOI 3
```

Sample Usage of Replace Commands.

4.10.3.1 MULTIPLE REPLACEMENTS WITHIN A LINE

Multiple string replacements in a single line are possible without executing a F command if the preceeding update instruction was an Input command or a Replace-string command. The resulting form will be displayed after each replacement, and the current line pointer will remain on the last line to be edited. Re-listing the modified line before an F command will display the current form rather than the modified form.

The intent of multiple replacements within a line is to minimize typing and buffer switching (the F command). If there are several elements of a line which you wish to change, you may change them one at a time, using the R command for each, without using the F command in between. On each use of the R command in this case, the command operates on the result of the last command. Only the first use of the R command operates on the original line. This means that if the X command is used, you move back to the original line, rather than the line as it was before the last use of the R command, because the last copy is not saved. In general, you can modify a line indefinitely.

If the replacement was a full-line replacement of the form R, carriage-return, followed by the prompt, followed by the text and a terminal carriage-return, the line may not be modified by a string replace until the buffers have been exchanged using the F command. The premise is that the X command can be used, followed by another replace. If this is not satisfactory, then the sequence, 'DE<I text<', will have the same result, and will allow replacements within the inserted line.

4.10.3.2 REPLACEMENT AFTER MULTIPLE-LINE REPLACEMENT

You may replace text in the last line of an Rn group using another R command without first flipping the buffers (the F command) in the same way it can be modified after a single-line replacement command. It is not possible to access lines prior to the last without using either the F command, which exchanges the buffers, or the X command, which cancels the Rn replace command.

4.10.3.3 MULTIPLE REPLACEMENTS AFTER THE MERGE COMMAND

It is possible to merge one or more lines of text into the current location in the text, and then modify the only or last line merged in using the multiple replace facility. Lines prior to the last can not be so modified for the reasons noted above. It is possible to do a lot of text manipulation very quickly using the merge, delete and replace commands.

4.10.3.4 CREATING NULL LINES - EDITOR

As discussed in the topic describing the Input command, the Replace command may be used to create null lines. This is accomplished by using the Input command to create lines each containing a fill character (such as an "."), and then prior to permanently filing the item replace each fill character with a null via a Replace command (such as R99/.//).

EXAMPLES:

```
                                Consider the following line.

084 The difference between the beginning and ending

                                To relace 2nd "the" with "any" :

C   [CR]
          1         2         3         4         5
12345678901234567890123456789012345678901234567890123456789012345

                                Useful aid for column identification.

R/the/any/24   [CR]             which will yield:

084 the difference between any beginning and ending

                                A column range example:

R/the/any/24-26   [CR]

084 the difference between any beginning and ending

                                Further unrelated examples:

R5/XYZ/123/15  [CR]             Replace first occurence of string
                                "XYZ" after column 15 for the
                                next 5 lines.

RU7/XX/77/20-50   [CR]          Replace all occurences of "XX"
                                between columns 20 to 50 for the
                                next 7 lines.
```

Sample usage of column specifications with the replace command.

4.11 ITEM MANIPULATING - EDITOR

Editor commands are provided for merging updates into the item, filing the item, deleting the item, or exiting an item.

4.11.1 "F" COMMAND - EDITOR

FORMAT:

```
F
```

The F command toggles the function of the EDITOR buffers. Updates are merged with the previously existing item, and the current line pointer is set to zero.

4.11.2 "FI" - FILE ITEM COMMAND : EDITOR

FORMAT:

```
FI{K}                          or

FI{K}{O} itemname              or

FI{K}{O}({DICT}filename {itemname}
```

The File Item command updates the edited item to the disc-file and returns control to TCL. When the item has been filed, the message "xxxx FILED" (where xxxx is the item name) is printed.

You may file the item currently being edited to either a different item in the current file, or to the same item name or to a different item name in a different file, by using the complex form of the "FI" command. Note the delimiter (space or left parenthesis) must immediately the "FI". Use a blank as a delimiter when only the itemname is specified. The default is the currently edited file. Any item-ids with embedded blanks may be enclosed in parenthesis. The DICT or filename, if present must immediately follow the left parenthesis, (no blanks). A copy of the edited item is generated to the designated file, and an updated version of the currently edited item is copied to the disc. Control is returned to TCL unless a selected list is in effect, in which case the next item is entered. The "K" option will cancel any selected list in effect and return control to TCL or a calling Proc. The "O" option will overwrite any item with the same name as the item we have instructed the Editor to generate, if it already exists in the designated file.

4.11.3 "FS" - FILE SAVE COMMAND : EDITOR

FORMAT:

```
FS                             or

FS{O} itemname                 or

FS{O}(filename {itemname}
```

The File Save command updates the edited item to the disc-file and returns control to the EDITOR. The current line pointer is set to zero.

You may file the item currently being edited to either a different item in the current file, or to the same item name or to a different item name in a different file, by using the extended syntax forms of the "FS" command. The "FS" command generates a copy of the item being edited to the designated file, updates the currently edited item, and returns control to the Editor. The "O" option would overwrite any pre-existing item in that designated file. Once again note the blank used as a delimiter with itemname only, and the need to put DICT or the filename immediately following the left parenthesis, which immediately follows the "FS" in extended forms of the command.

4.11.4 "FD" - FILE DELETE ITEM : EDITOR

FORMAT:

FD{K}

The File Delete command deletes the item from the disc-file and returns control to TCL.

When the item has been deleted, the message "xxxx DELETED" (where xxxx is the item name) is printed. You can not FD any item other than the one which you are currently editing. Massive use of the FD can be accomplished with the DELETE verb (PROC) or by the use of a prestore command. The delete verb will be faster.

The "K" option returns control to TCL or a calling Proc, before any remaining selected items need be edited.

4.11.5 "EX" - EXIT COMMAND : EDITOR

FORMAT:

EX{K}

The Exit command terminates the EDITOR session and returns control to TCL. The item being edited will not be updated to the disc-file. Upon exit, the message "'ITEM-ID' EXITED" is printed.

The user should again note that if multiple items were specified in the EDIT verb at the TCL level, then any of the above commands which ordinarily return control to TCL will instead return control to the EDITOR to edit the next item which was specified.

The purpose of the "EXK" command is to exit from just such a situation, and to cause the editing process to proceed to TCL or the PROC which called the EDITor. The exit process will not recognize a lowercase 'k'.

NOTES: In general, anything which the EDITOR either does not understand or of which the EDITOR disapproves will result in CMND? Error message when the FI, FS, or EX processes are involved.

```
* >ED AFILE ABC  [CR]
  TOP
* .P  [CR]
  001 AAAAAAAAA  --| <-------- This is what item ABC looks like.
  002 12121212   --|
  EOI 2
* .DE   [CR] <---------------- Delete command (deletes line 2).
  EOI 2
* .F  [CR] <------------------ F command (toggles buffers).
  TOP
* .P  [CR]
  001  AAAAAAAAA <------------ Here is item ABC after deletion.
  EOI 1
* .EX  [CR] <----------------- Exit command (returns control to TCL
  'ABC' EXITED                        but does not file updated item).

* >ED AFILE ABC  [CR]
  TOP
* .P   [CR]
  001 AAAAAAAAA  --| <-------- Item ABC still contains 2 lines since
  002 12121212   --|           Exit command above did not file updated
  EOI 2                        item.
* .DE   [CR] <---------------- Delete command (deletes line 2).
  EOI 2
* .FI  [CR] <----------------- File Item command (files item and
  'ABC' FILED.                Returns control to TCL).
* >ED AFILE ABC  [CR]
   TOP
* .P   [CR]
  001 AAAAAAAAA <------------- Here is item ABC (note that line 2 is
  EOI 1                        now permanently deleted).
* .FS  [CR] <----------------- File Save command (files item and
  TOP                          returns control to EDITOR).
* .FD  [CR] <----------------- File Delete command (deletes item and
  'ABC' DELETED.               Returns control to TCL).

  > <------------------------- TCL verb awaited.
```

Sample Usage of Item Manipulating commands.

4.12 FORMATTING COMMANDS : EDITOR

The Editor Formatting commands aid the user with some very useful tools to assist in handling edited items.

4.12.1 "S" - SUPPRESSION COMMAND : EDITOR

FORMAT:

S

The "S" command is used to suppress Editor line numbers. Entry of an "S" command acts as an alternate-action toggle switch. The Editor will respond with "SUPPRESS ON" or "SUPPRESS OFF" accordingly.

When the "S" command is used with Assembly Language programs and the "AS" command (standard assembly listing format), it takes on an additional feature. If the "AS" command is in effect, ("AS-ON") the "S" command causes the suppression of the Object Code. With "AS" disabled ("AS-OFF"), the "S" command suppresses line numbers as with a non-assembler data item.

The suppress feature may also be enabled by using the " (S " option with an edit command.

4.12.2 "TB" - TAB COMMAND : EDITOR

FORMAT:

TB n,n,n, ...

The n's consist of up to 15 Tab Settings (in ascending order), seperated by commas.

Tabbing is invoked whenever the EDITOR is in the Input Enviroment and a control-I or on some terminals a TAB key, is pressed. The TAB key will cause a series of blanks to be output, thus moving the cursor (or printer) to the next specified tab stop. A backspace and cancel will backspace over tabs.

Tabs set by the EDITOR are identical to those set by the external TAB command.

4.12.3 "Z" - ZONE COMMAND : EDITOR

"Z" - ZONE COMMAND : EDITOR

FORMAT:

Z{p{-q}}

This command sets print column limits for listing output of lines via the List command (i.e., only column positions p through q of each line will be listed). If p and q are omitted, the zone is reset so that the entire line will be listed on output. If q<p, q=p is assumed. Setting a zone does not affect the search for a "string" in the Locate, Delete or Replace commands.

EXAMPLES:

```
* >ED FN5 XX  [CR]
  NEW ITEM <----------------- This is a new item.
  TOP
* .TB 9,18  [CR] <----------- Tab command (sets 2 tab stops).
* .I  [CR] <----------------- Input command.
* 001 ABC  [CR]
* 002 ABCD    EF  [CR] <----- Lines being input; note that for line
* 003 123456789  [CR]        2 a control-I (which does not print)
* 004 [CR]                   was entered after "ABCD" causing the
  TOP                        EDITOR to tab over to the 1st stop.
* .Z2-3  [CR] <-------------- Zone commands (limits listing output
* .P    [CR]                 to columns 2 thru 3).
  001 BC  --
  002 BC   | <--------------- Only columns 2 thru 3 are listed.
  003 23  --
  EOI 3
* .T  [CR]
  TOP
* .Z  [CR] <----------------- Zone command (restores full line).
* .S  [CR] <----------------- Suppress command (suppresses line
* .P    [CR]                 numbers).
  ABC          --
  ABCD     EF  | <----------- Line numbers are suppressed.
  123456789   --
  EOI 3
* .S  [CR] <----------------- Suppress command (restores line numbers).
* .P    [CR]
  TOP
  001 ABC         --
  002 ABCD     EF | <-------- Line numbers are listed.
  003 123456789  --
  EOI 3
  .
```

Sample Usage of Editor Formatting Commands.

4.13 ASSEMBLY FORMATTING : EDITOR

The Assembly formatting commands are invaluable features when using Assembly Language Code.

4.13.1 "AS" - ASSEMBLY FORMAT COMMAND : EDITOR

FORMAT:

```
AS
```

The "AS" command is used to format assembly code source programs in the standard assembly listing format. The "AS" command acts as an alternate-action toggle switch to either format assembly code source program lines in the assembly listing format, or to revert to unformatted form.

The EDITOR will respond with the message "ASM-ON" or "ASM-OFF", depending on the previous state.

This mode may also be turned on when entering the EDITOR by using the "(A)" option on the EDIT command.

Assembly-code source programs contain the assembled object code and macro expansions along with the original source text. If displayed in nomal form, a line might look like:

```
007 L00P STORE D1 SAVE ACCUMALATOR\01B A00499
```

If the "AS" mode is set on, the same line will be displayed as:

```
007 01B A00499          LOOP STORE D1          SAVE ACCUMALATOR
...object code...     ...source code...     ...comment field...
```

This display format does not affect the search columns in Locate, Delete or Replace commands, which use the internal (unformatted) form.

When the "AS mode is on, the "S" (suppress) command will act to suppress object-code, not line numbers.

4.13.2 "M" - MACRO EXPANSION COMMAND : EDITOR

FORMAT:

```
M
```

When in the "AS" mode the "M" command will cause macros to be expanded. A Macro Expansion is generally a line of code which breaks down and is defined by one or more lower machine level instructions. It is normally off. Execution of the M command will cause the EDITOR to respond with the message "MACRO-ON" or "MACRO-OFF", depending on the previous state.

EXAMPLES:

```
*  ED SM TERMIO  [CR]
  .TOP
* .L3  [CR]
  001  FRAME 006] FRM: 006\001 7FF00006] ORG 1\001
  002 *SYSTEM*UTILITY
  003 *30SEP84
* .AS  [CR] <------------------------------ Turns assembly formatting
  ASM-ON                                    mode ON.
* .T  [CR]
  TOP
* .L3  [CR]
  001 001 7FF00006              FRAME 006
      001
  002                  *SYSTEM*UTILITY
  003                  *30SEP84
* .S  [CR] <------------------------------- Suppress object code.
  SUPPRESS ON
* .T  [CR]
* .L3  [CR]
  001 FRAME 006
  002 *SYSTEM*UTILITY
  003 *30SEP84
* .S  [CR] <------------------------------- Clear Suppress mode.
  SUPPRESS OFF
* .AS  [CR] <------------------------------ Clear Assembly formatting
  ASM-OFF                                   mode.
* .T  [CR]
  TOP
* .L3  [CR]
   FRAME 006] FRM: 006\001 7FF0006] ORG 1\001
   *SYSTEM*UTILITY
   *30SEP84
   .
```

Sample Usage of Assembly Formatting commands.

4.14 MISCELLANEOUS COMMANDS : EDITOR

There are a few miscellaneous Editor commands which allow for some helpful features in editing items.

4.14.1 'X' CANCEL COMMAND : EDITOR

FORMAT:

X{F}

The "X" command deletes the effect of the last Input, Insert, Delete, or Replace command that was issued. This is useful if one of these commands has been erroneously entered.

When the effect of the update command has been deleted, the message "L n" will be printed (where is the line number of the line whose update was deleted). The X command will not work after multiple string replacements within a single line.

The XF command will reverse the effect of all updates executed since the last buffer exchange (F command).

4.14.2 '?' CURRENT LINE COMMAND : EDITOR

FORMAT:

?

When a Current Line command '?' is entered, the editor will respond with the item-id and the current line number, of the item being edited.

4.14.3 'S?' ITEM SIZE COMMAND : EDITOR

FORMAT:

S?

The size of the item being edited may be discovered with the S? Command. It will output the total size of the item for file purposes.

4.14.4 '^' WILDCARD TOGGLE COMMAND : EDITOR

FORMAT:

```
^
```

The "^" command acts as an alternate-action toggle switch to turn off or on the special effect of the "^" character within a "string".

The EDITOR will respond with the message "/^\ ON" or "/^\ OFF".

4.14.5 'C' COLUMNAR POSITIONS COMMAND : EDITOR

FORMAT:

```
C
```

The "C" command will print out a list of column numbers so that the user can readily determine the columnar position of data in a line. This is particularly helpful when editing fixed-field data, or RUNOFF documentation.

4.14.6 UNPRINTABLE CHARACTERS

Characters which are unprintable include the control characters, between X'00' and X'1F', inclusive. The Editor marks control characters by inserting a period, '.', where the control character stands in the text line. It does not indicate what the character is, however. It may then be removed by replacing a unique string which includes the control character with the string of your choice. The control character should be marked with an ^ in the first string in the replace.

4.15 'Pn' PRESTORE COMMAND - EDITOR

The prestore facility allows the storage of up to 10 strings of Editor commands, and the execution of the string by using the name of the string as the command.

The allowable string names are P0, P1, ... , P9. There are therefore a maximum of ten prestored commands available at any one time. Further, each prestored command is allocated 100 bytes, so that, if one wishes to generate a prestored command which exceeds 100 bytes, simply do not initialize the command whose name is ordinally next. (If P1 is 150 bytes, do not use P2.)

4.15.1 DEFINING PRESTORE COMMANDS - EDITOR

Assume you are currently editing an item and the system is awaiting input at the " . " (Editor Prompt). In order to create a prestored command, type in the name of the prestored command, P0, P1, ..., P9, followed by a space, followed by the first command to be executed, followed by the prestore command delimiter, which is a start buffer mark (X'FB'), and which may be input by typing CONTROL-[(control-left-square-bracket) or ESCAPE (esc), followed by the next command, and so on. Any valid command is usable, including prestore command names.

P0 L22	This is loaded when you enter the EDITor. It has the following synonym:
P L22	This allows the traditional L22< to be done by P<, which is generally convenient, since it is next to the carriage-return key.
P1 R100/DOG/CAT[F[R100/dog/cat[FI	This has the effect of changing dogs to cats in the first hundred lines of text.

Creating simple prestores.

4.15.1.1 PRESTORE COMMAND - DEFAULTS

In the above examples, note first that 'P' is a synonym for 'P0'. It is automatically loaded with 'L22' at entry to the EDITor. P1 through P9 are null at entry. Executing them will cause a CMND? Response. All prestores created since the entry to EDITor are retained until the EDIT verb is exited. Any of them may be changed by creating another prestore command string with the same name. The prestores persist from item to item, whether the EDITor is using an explicit item list, a selected list, or the whole file.

4.15.2 REPEATING PRESTORE COMMANDS

If one is going to use a prestore command for a repetitive task, it may either be activated each time it is to be used, or it may call itself, at which time its termination conditions must be considered. A prestore command which calls itself will terminate only when it runs out of items to process. This means that a prestore which calls itself must have an EX, FI, or FD is the command string. If it does not have such an item iteration command in the string, it will loop indefinitely in the current item. The only exit from this condition is a BREAK-and-END. The primary use of the prestore calling itself is to manipulate many items with a single instruction string initiated once. It is particularly useful for searching for specified strings in text files and replacing them as necessary. The following example searches a BP (BASIC program) file for the name GENERAL.LEDGER.

```
ED BP *<                              Edit the file.
ITEMNAME                              The first item.
TOP                                   Standard mark.
.P1 L500/GENERAL.LEDGER[EX[P1         Define the search.
.P1                                   Initiate the run.

                                      At this point the EDITor will
                                      exhibit all lines in the current
                                      item with the desired string,
                                      and then display
EOI nnn                               the number of lines in the item
'ITEMNAME' EXITED                     and the name of the item exited.
NEWITEMNAME                           The name of the next item.
TOP                                   The top mark.
                                      All the lines with the string, if
                                      any, and so on, until the list is
                                      exhausted, at which time the
                                      process will return to TCL.
```

A prestore command calling itself.

The same maneuver may be executed to the printer by appending the (P option to the EDIT verb. In this case, all information which would have been displayed on the terminal will be sent to the printer.

4.15.3 DISPLAYING PRESTORE COMMANDS

It is possible to display all currently initialized prestore commands by using the PD (Prestore Display) command.

In the above example, if the PD command was executed before the P1 command, the following would result:

```
PD                                  The prestore display command
                                    will yeild:
P0  L22                             The default, and
P1  L500/GENERAL.LEDGER[EX[P1       which is the command
                                    defined above.
```

The PD command.

4.15.3.1 PRESTORES IN PROCS

It is possible to create the desired prestored command strings in PROCS in the same manner that instructions are sent to the various processors from a PROC.

```
PQ                                  The PROC definition.
HED BP *                            The verb.
STON                                Turn the stack on.
HP1 L500/GENERAL.LEDGER<            Specify the prestore.
HP1<                                Execute the prestore.
P                                   Execute the verb.
```

Defining a prestore in a PROC.

The example above assumes that a list is in existence. The verb activation may include an explicit item list or specify the whole file using the conventional asterisk. On entry to the first item from the EDIT verb, the prestore is automatically set up, and is available for use. All ten prestores may be initialized this way, allowing the development of powerful customized EDITor commands.

```
* >ED CARS TEN  [CR]
  TOP
* .L99  [CR]
  001 A1234       --
  002 C1234         | <---------- This is what item TEN looks like.
  003 Xxxxx1234     |
  004 ABCDE1234   --
  EOI 4
* .G1
  001 A1234
* .DE  [CR] <------------------- Delete command (deletes line 1).
* .X  [CR] <-------------------- X command (cancels effect of Delete
                                 command).
  L 1 <------------------------- Message indicates that update on line
* .F  [CR]                       1 was cancelled.
  TOP
* .L99  [CR]
  001 A1234 <------------------- Line 1 was not deleted.
  002 C1234
  003 XXXXX1234
  004 ABCDE1234
  EOI 4
* .?  [CR] <-------------------- Current Line command.
  L 4 <------------------------- Current line is line 4.
* .P DE"1234"6-9  [CR] <-------- Prestore command (prestores Delete
* .T  [CR]                       command).
  TOP
* .P  [CR] <-------------------- Prestore Call command (calls Delete
                                 into effect).
  003 XXXXX1234 <--------------- Line 3 deleted.
* .F  [CR]
  TOP
* .P  [CR] <-------------------- Prestore Call command.
  003 ABCDE1234 <--------------- New line 3 deleted.
  EOI 3
* .F  [CR]
  TOP
* .L99  [CR]
  001 A1234  --| <-------------- Here is item TEN after deletions.
  002 C1234  --|
  EOI 2
```

Sample Usage of X, Current Line, and Prestore Commands

4.16 EDITOR MESSAGES

| This appendix presents a list of the messages output by the EDITOR. |

MESSAGE	DESCRIPTION	EXAMPLE CAUSING ERROR
CMND?	Illegal EDITOR command.	XYZ
STRING?	Illegal specification, or missing string (e.g., required string missing for Merge; second string missing for Replace). This message may also occur as a result of an illegal numeric parameter specification, which causes a part of the numeric parameter to appear as if it were a string.	ME 10 R5/ABC/
COL#?	Illegal characters follow the recognized end of the command, or illegal format for a column-number limit specification, or non-numeric characters used for p and q in Locate, Replace, Delete or Merge Commands.	L.10.23. R/ABC/DEF/X R/M/DICT/MD L,SMITH,JOHN,
SEQN?	Out-of-sequence update; updating must be done in an ascending line number sequence until an F command is entered.	
EOI m	End-of-item reached at line m.	
TOP	Top-of-item (line 0) reached.	
L n	Specifies that n is the current line number or specifies that update action on line N was deleted via and X command.	
NOT ON FILE	Item specified in Merge command is not on the disc-file.	
'xxx' EXITED	Editor exited via EX command.	
'xxx' DELETED	Item with name xxx has been deleted from the disc-file.	
'xxx' FILED	Item with name xxx has been updated to the disc-file.	

SECTION 5

THE ICON/PICK PROC LANGUAGE

ICON®

Chapter 5

PROC LANGUAGE

THE PICK SYSTEM

USER MANUAL

5.1 THE PROC PROCESSOR

| This chapter describes the PROC (stored procedure) processor. |

The system allows the user to prestore a complex sequence of Terminal Control Language (TCL) operations (and associated processor operations) which can then be invoked by a single word command. Any sequence of operations which can be executed at the TCL level can also be prestored via the PROC processor. This prestored sequence of operations (called PROC) is executed interpretively by the PROC processor and therefore requires no compilation phase.

The PROC processor has the following features:

- Four variable length I/O buffers
- Parameter passing between buffers
- Interactive terminal prompting
- Extensive I/O and buffer control commands
- Conditional and unconditional branching
- Relational character testing
- Pattern matching
- Free-field and fixed-field character manipulation
- Optional command labels
- User-defined subroutine linkage
- Inter-PROC linkage

A PROC provides a means to prestore a highly complex sequence of operations which can then be invoked from the terminal by a single command.

The usage of the PROC processor is quite similar to the use of a Job Control Language (JCL) in some large-scale computer systems. The PROC language in the Pick Computer System, however, is more powerful since it has conditional capabilities, and can be used to interactively prompt the terminal user. Additionally, a PROC can test and verify input data as they are entered from the terminal keyboard.

A PROC is stored as an item in a dictionary or data file. The first attribute value (first line) of a PROC is always the code PQ. This specifies to the system that what follows is to be executed by the PROC processor. All subsequent attribute values contain PROC statements that serve to generate TCL commands or insert parameters into a buffer for interactive processors (such as the EDITOR). PROC statements consist of an optional numeric label, a one or two-character command, and optional command arguments.

PROC's operate on four input/output buffers; the primary input buffer, the secondary input buffer, the primary output buffer, and the secondary output buffer (called the stack). Essentially, the function of a PROC is to move data from either input buffer to either output buffer, thus forming the desired TCL and processor commands. At any given time, one of the input buffers is specified as the "currently active" input buffer, while one of the output buffers is specified as the "currently active" output buffer. Buffers are selected as "currently active" via certain PROC commands. Thus, when moving data between the buffers, the source of the transfer will be the currently active input buffer, while the destination of the transfer will be the currently active output buffer.

The primary input buffer contains the PROC name and any optional arguments, exactly as they were entered when the PROC was invoked. The primary output buffer is used to build the command which will ultimately be submitted at the TCL level for processing.

The secondary input buffer contains data subsequently input by the user in response to an IN command. Usually the data in this buffer will be tested for correctness and then moved to the secondary output buffer (the stack). When all desired data has been moved to the secondary output buffer, control will be passed to the primary output buffer via a P or PP command. The command which resides in the primary output buffer will be executed at the TCL level and the data in the secondary output buffer (if any) will be used to feed processors such as ACCESS or EDITOR. When the process is completed, control returns to the PROC at which time new data may be moved to the output buffers.

Once a PROC is invoked, it remains in control until it terminates. When the PROC temporarily relinquishes control to a processor such as the EDITOR or a user-supplied subroutine, it functionally remains in control since an exit from the called processor returns control to the PROC. TCL only regains control when the PROC is terminated explicitly, or when all of the lines in the PROC have been exhausted.

COMMAND	BRIEF DESCRIPTION
A	Moves data argument from input to output buffers.
B	Backs up input pointer.
BO	Backs up output pointer.
C	Specifies comment.
D	Display either input buffer to terminal.
F	Moves input pointer forward.
GO	Unconditionally transfers control.
H	Moves text string to either output buffer.
IF	Conditionally executes specified command.
IH	Moves text string to either input buffer.
IP	Inputs from terminal to either input buffer.
IS	Inputs from terminal to secondary input buffer.
IT	Inputs from tape to primary input buffer.
O	Outputs text string to terminal.
P	Causes execution of PROC.
PP	Displays content of output buffers and executes PROC.
PW	As above, waits for user response before proceeding.
PH	As in P, but suppresses all terminal output for the verb.
PX	As in P, will return to TCL after processing, not to PROC.
RI	Clears (resets) input buffers.
RO	Clears (resets) output buffers.
S	Sets position of input pointer and optionally selects primary input buffer.
SP	Selects primary input buffer.
SS	Selects secondary input buffer.
ST ON	Selects secondary output buffer (stack on).
ST OFF	Selects primary output buffer (stack off).
T	Provides formatted terminal output.
U	Exits to user-defined subroutine.
X	Exits back to TCL level, or calling PROC.
+	Adds decimal number to a parameter in input buffer.
-	Subtracts decimal number from a parameter in input buffer.
()	Transfers control to another PROC.
[]	Subroutine call, local or to another PROC.

Summary of PROC Commands.

5.3 AN INTRODUCTION TO PROC'S

An integral part of the Pick Computer System is the ability to define stored procedures called PROC'S.

A PROC provides the applications programmer a means of creating a sequence of operations which can then be invoked from the terminal by a one word command. Any operation that can be executed by the Terminal Control Language can be performed in a PROC. This usage of a PROC is quite similar to the use of a Job Control Language (JCL) in some computer systems. The PROC language in the Pick Computer System, however, is more powerful since it has conditional capabilities, and can be used to interactively prompt the terminal user. Additionally, a PROC can test and verify input data as they are entered from the terminal keyboard.

A PROC is executed interpretively by the PROC processor and therefore requires no compilation phase. A PROC stored as an item in the user's Master Dictionary (M/DICT) is executed in the TCL environment by typing the item-id of the PROC, any optional arguments, and a carriage return.

While a PROC must exist in the Master Dictionary, the actual body of the PROC may be within the same item, or it may be stored as an item in any dictionary or data file. The first attribute (first line) of a PROC is always the code PQ. This specifies to the system that what follows is to be executed by the PROC processor. All subsequent attribute values contain PROC statements that serve to generate TCL commands or insert parameters into a buffer for the interactive processors, such as the EDITOR or the BATCH processor. PROC statements consist of an optional numeric label, a one or two character command, and optional command arguments. PROC's are created using the EDITOR.

The ability to interactively prompt input data from the user (and subsequently verify these data) is demonstrated. The PROC then prompts the user for the required data. The PROC could then, for example, store these data in a buffer which would then be passed to another processor to update the file.

Once a PROC is invoked, it remains in control until it terminates. When the PROC temporarily relinquishes control to a processer such as the EDITOR, PICK/BASIC, etc., or a user-supplied subroutine, it fuctionally remains in control since an exit from the called processor returns control to the PROC. TCL only regains control when the PROC is terminated explicitly, or when all of the lines in the PROC have been exhausted.

```
>LISTU [CR]

 CH# PCBF NAME........ TIME... DATE.... LOCATION.........

  00 0200 SP          08:00AM 01/01/78 Channel 0
  02 0240 CM          09:10AM 01/01/78 Channel 2
  03 0260 LC          07:30AM 01/01/78 Channel 3
  04 0280 JP          10:14AM 01/01/78 Channel 4
 *06 02C0 SAL         08:35AM 01/01/78 Channel 6
  10 0340 JET         09:00AM 01/01/78 Channel 10
```

Sample PROC Execution.

```
>LISTDICTS POLICY [CR]

POLICY.............. D/CODE.. A/AMC.. V/CONV..... V/TYP V/MAX

AUDIT-PERIOD        A        01                   L     4
POLICY-PERIOD-FROM  A        02       D           L     10
POLICY-PERIOD-TO    A        03       D           L     11
EXPIRES             A        04       D           L     12
```

Sample PROC Execution. (Parameter Passing)

```
>ENTER-DATA [CR]

PART-NUMBER              = 3215-19 [CR]
DESCRIPTION              = TRANSISTOR [CR]
QUANTITY                 = FIFTY [CR]

ERROR:NUMERIC DATA ONLY!!

QUANTITY                 = 50 [CR]
```

Sample PROC Execution. (Interactive Prompting)

Operations specified within a PROC involve the movement of data from either of two input buffers (data storage areas) to either of two output buffers.

PROC utilize four input/output buffers: the primary input buffer, the secondary input buffer, the primary output buffer, and the secondary output buffer (called the stack). The general relationship of these buffers is illustrated in the first example. Essentially, the function of a PROC is to move data from either input buffer to either output buffer, thus forming the desired TCL and processor commands. At any given time, one of the input buffers is specified as the "currently active" input buffer, while one of the output buffers is specified as the "currently active" output buffer. Buffers are selected as "currently active" via certain PROC commands (these commands are discussed in detail in the remaining topics of this section). Thus, when moving data between the buffers, the source of transfer is the currently active input buffer, while the destination of the transfer is the currently active output buffer.

The primary input buffer contains the PROC name and any optional arguments, exactly as they were entered when the PROC was invoked. The contents of this buffer remain the same througout execution of the PROC unless explicitly modified by an IP, IT, IH, RI, Plus or Minus command.

The primary output buffer builds the single command which ultimately is submitted at the TCL level for processing. Any command which can be executed via the terminal at the TCL level can also be constructed and executed via a PROC.

The secondary input buffer contains data subsequently input by the user in response to an IS command. The data in this buffer are volatile and are overwritten by subsequent IS commands. Usually the data in this buffer is tested for correctness and then moved to the secondary output buffer (the stack).

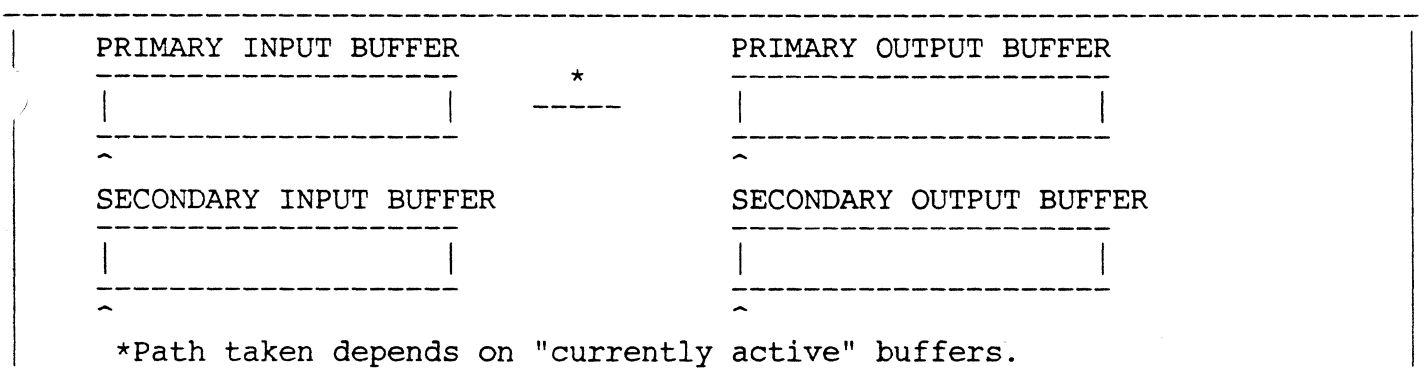
The secondary input buffer is now loaded with data from several system processors, most notably the spooler. Information such as last hold file entry number is placed into this buffer. More information on this can be found in the spooler documentation in the PERIPHERALS manual. The user should note that the secondary input buffer is a very temporary entity and that if its contents are to be used, this should be done immediately subsequent to the execution of the processor which loaded the buffer.

The secondary output buffer ("stack") contains data that is to be used by the processor called by the PROC generated TCL statement. Zero or more lines may be stored in the stack. Each request for terminal input by the called process or (for example each INPUT statement in BASIC) will be satisfied with a line of data from the stack. In the event that the called processor requests more data than exists in the stack, data will be requested from the terminal from that point onwards.

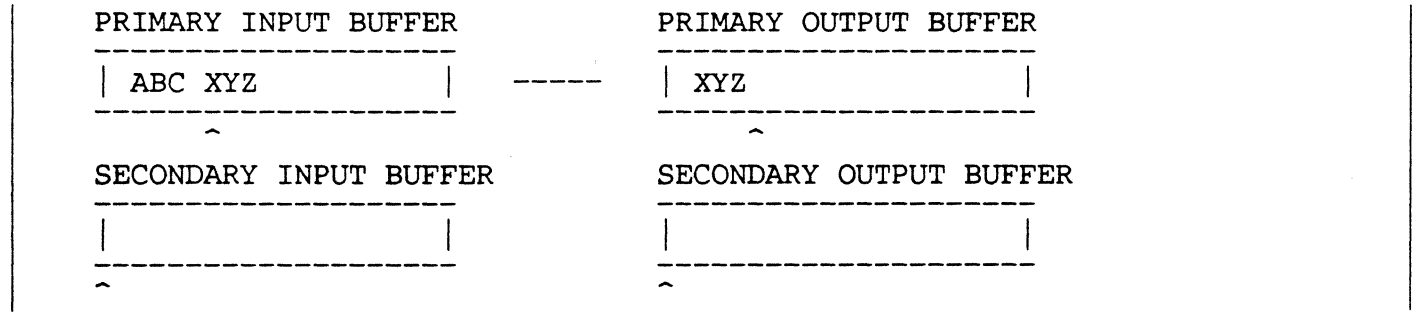
Note that each line of data in the secondary output buffer must be terminated by a carriage return which is explicitly placed in the stack via an H command (refer to the topic describing that command). This is not the case with the primary output buffer; a carriage return is automatically placed at the end of the TCL command in the primary output buffer upon execution of that buffer via the P, PW, PH, PX or PP command.

When all desired data have been moved to the output buffers, control is passed to TCL via a P, PH, PX, PW or PP command. The command which resides in the primary output buffer is executed at the TCL level and the data in the secondary output buffer (if any) is used to feed processors such as PICK/BASIC or the EDITOR. When the process is completed, control returns to the PROC, at which time new data may be moved to the output buffers.

Moving data between the buffers is done in terms of "parameters". A parameter is defined as a string of characters (residing in one of the buffers) which is surrounded by blanks or surrounded by quotes. To keep track of the parameters, each buffer has a pointer which points to the "current" position of that buffer. These pointers are depicted in the buffer diagrams as small arrows placed beneath the buffer. As a general illustration of this concept, consider the sample situation illustrated in the second example. Here the PROC has been invoked by the characters ABC XYZ, which are then automatically placed in the primary input buffer. PROC commands have then been processed which position the input pointer of the primary input buffer to the second parameter (XYZ), and then subsequently move that parameter to the primary output buffer (i.e., the currently active buffers are the primary input buffer and the primary output buffer).



PROC Input/Output Buffers.



Sample Inter-Buffer Transfer With Both Primary Buffers Currently Active

| A PROC consists of any number of PROC commands, one command per line. |

The first line (attribute) of a PROC must contain the code PQ. This identifies the item as a PROC. The remaining lines in the PROC may contain any valid PROC commands. There is no limit to the number of lines in a PROC. However, each line may contain only one command, and each command must begin in column position one of the line.

PROC commands are listed in alphabetical order in the example. A complete description of each command type is presented in the remaining topics within this section.

Any PROC command may optionally be preceded by a numeric label. Such a label serves to uniquely identify its associated PROC command for purposes of branching or looping within the PROC. Labels may consist of any number of numeric characters (e.g., 5, 999, 72, etc.). When a label is used , the PROC command must begin exactly one blank beyond the label. For example:

```
1 GO 5
23 A
99 IF A = ABC GO 3
2 ST ON
```

Only the first occurence of the label is used as the destination of any control transfers; i.e., no check is made for erroneous duplicate labels!

As an introductory example to PROC commands, consider the following PROC stored as item 'DISPLAY' in the user's MD:

```
001  PQ
002  HLIST ONLY
003  A2
004  P
```

Assume that the user types in the following:

```
>DISPLAY INVENTORY  [CR]
```

This input invokes the above PROC and places the words DISPLAY INVENTORY in the primary input buffer. The second line of the above PROC is an H command which causes the text LIST ONLY to be placed in the primary output buffer. The third line is an A command which picks up the second word (parameter) in the primary input buffer and places it in the primary output buffer. Thus the primary output buffer contains the words LIST ONLY INVENTORY. The last line of the PROC is a P command which submits the content of the primary output buffer to TCL for processing (i.e., LIST ONLY INVENTORY is an ACCESS sentence which causes the item-ID's of the INVENTORY file to be listed; refer to the ACCESS Manual).

COMMAND	BRIEF DESCRIPTION
A	Moves data from input to output buffers.
B	Backs up input pointer.
BO	Backs up output pointer.
C	Specifies comment.
D	Outputs from either input buffer to terminal.
F	Moves input pointer forward.
G or GO	Unconditionally transfers control.
H	Moves text string to either output buffer.
IF	Conditionally executes specified command.
IH	Moves text string to either input buffer.
IP	Inputs from terminal to either input buffer.
IS	Inputs from terminal to secondary input buffer.
IT	Inputs from tape label to primary input buffer.
O	Outputs text string to terminal.
P	Causes execution of a PROC.
PP	Displays contents of output buffers and executes PROC.
PW	As above, waits for user response before proceeding.
PH	As above but suppresses all terminal output for the verb.
PX	As in P, will return to TCL after processing, not to PROC.
RI	Clears (resets) input buffer.
RO	Clears (resets) output buffer.
S	Positions input pointer.
SP	Selects primary input buffer.
SS	Selects secondary input buffer.
STON	Selects secondary output buffer (stack).
STOFF	Selects primary output buffer.
T	Provides formatted Terminal output (Cursor Control).
U	Exits to user-defined subroutine.
X	Exits back to TCL level, or calling PROC.
+,-	Adds, subtracts decimal number to parameter in input buffer.
()	Links to another PROC.
[]	Subroutine call, local or to another PROC.

Summary of PROC commands.

The SP and SS commands select the primary or secondary input buffer, respectively, and set the input pointer at the beginning of the buffer. The STON will turn the stack on while the STOFF will turn the stack off.

The input buffers receive data from the terminal and store it so that it may be transferred to the output buffers. Only one of the two input buffers is "currently active". The SP and SS commands are used to select one or the other input buffer.

At the initiation of a PROC the primary input buffer is automatically selected, and the buffer-pointer is set to the start of the input buffer, which contains the name by which the PROC was called from TCL. After the execute-primary-output-buffer command (P, PH, PX, PP, or PW) the primary input buffer is selected, and the pointer set to the beginning of the buffer on return of control to the PROC from TCL. The contents of the primary input buffer are not disturbed, however.

The general form of the SP command is:

SP

It selects the primary input buffer and sets the input pointer at the beginning of the buffer.

The general form of the SS command is:

SS

It selects the secondary input buffer and sets the input pointer at the beginning of the buffer.

Note that the IS command will also select the secondary input buffer.

The primary output buffer is used to store one TCL statement that is eventually executed by a P, PH, PX, PP or PW command. The secondary output buffer (stack), is used to store zero or more lines of data to satisfy terminal input requests by the processor invoked by the above mentioned TCL statement. Note that the "stack" is a first-in, first-out queue.

Only one of the two output buffers is "currently active". The STON or STOFF commands are used to select one or the other output buffers. Upon initial entry to a PROC, the stack is off.

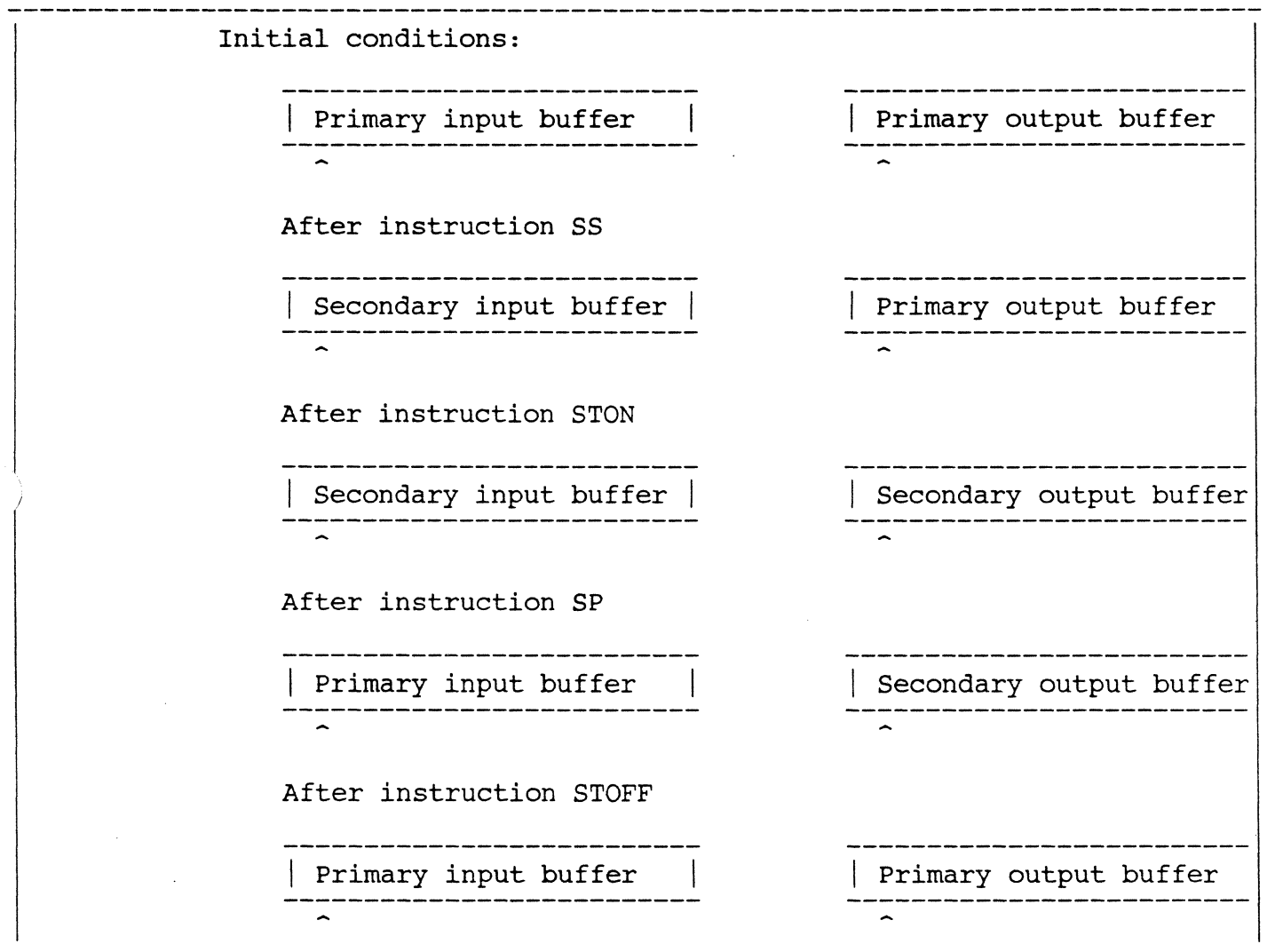
The STON command selects the secondary output buffer (the stack) as the currently active output buffer (i.e., turns the stack on). Its general form is:

STON or ST ON

The STOFF Command selects the primary output buffer as the currently active output buffer (i.e., turns the stack off). Its general form is:

STOFF or ST OFF

When the stack is on, all data picked up by the A command are moved to the secondary output buffer. When the stack is off, these data are moved to the primary output buffer. The stack may be turned on or off at any point within the PROC. The example below shows the results of these instructions. The pointers indicate currently active buffers in each case.



Sample usage of SS, SP, STON, STOFF Commands.

The S command positions the input pointer and/or selects the primary input buffer as the currently active input buffer. The F and B commands move the input pointer forward or backward one parameter, respectively. The BO command moves the output pointer backward one parameter.

The S command positions the input pointer in the currently active input buffer. This command may be used in the following general form:

Sp

Sp moves the input pointer to the p'th parameter of the currently active input buffer, where the parameters are seperated by blanks or enclosed in single quotes. If there is no pth parameter, the pointer is set to the end of the input buffer. S0 or S1 will set the pointer to the beginning of the buffer.

The F command causes the input pointer for the currently active input buffer to move forward one parameter. If the input buffer pointer is currently at the end of the buffer, this command has no effect.

The general form of the F command is as follows:

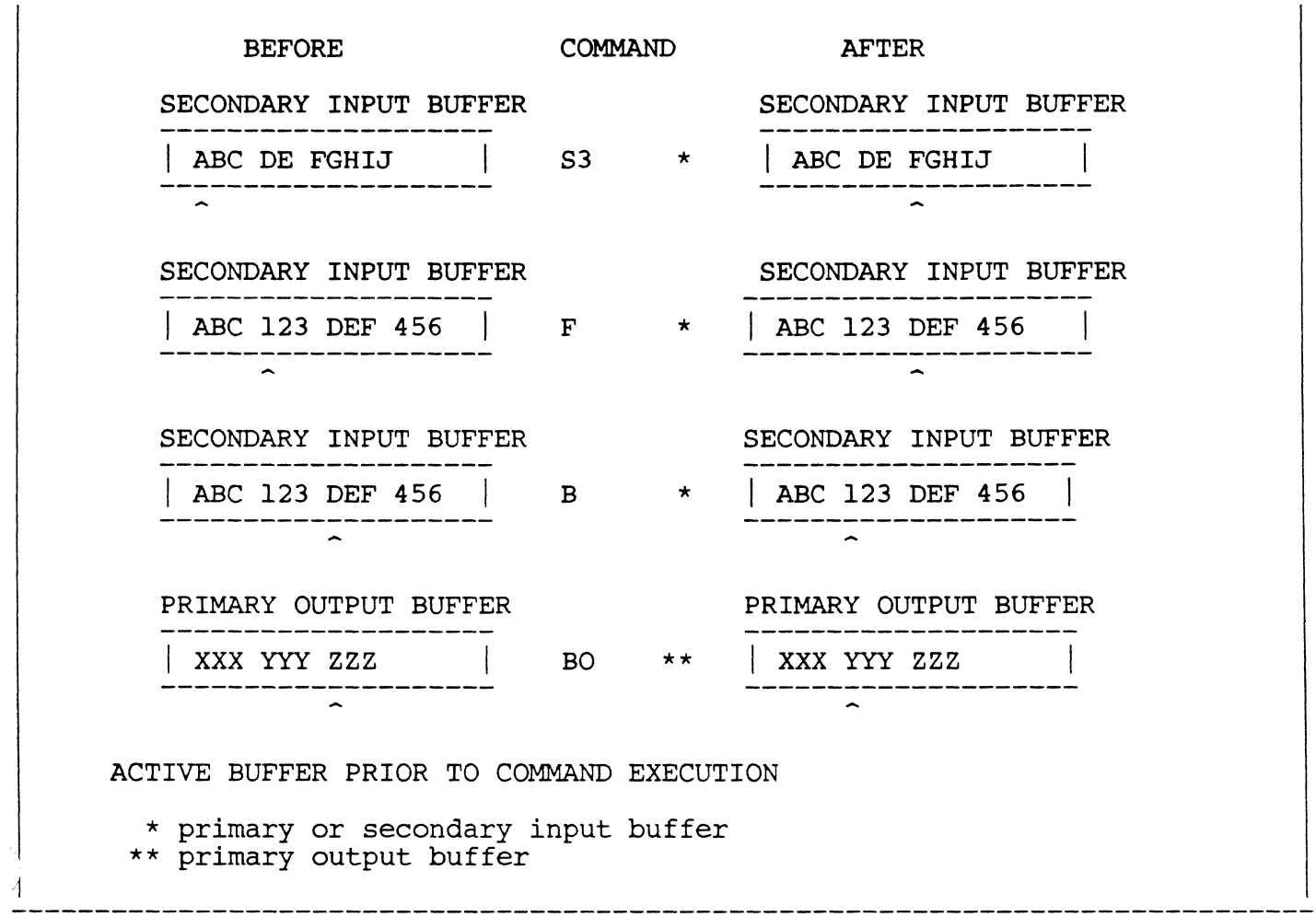
F

The B command causes the input pointer for the currently active input buffer to move backward one parameter. If the input buffer pointer is currently at the beginning of the buffer, this command has no effect. The general form of the B command is as follows:

B

The BO command causes the output pointer for the current output buffer to move backward one parameter. If the output buffer pointer is currently at the beginning of the buffer, this command has no effect. The general form of the BO command is as follws:

BO



Sample usage of S, F, B, BO Commands.

The A command is used to move a parameter from the input buffer to the output buffer. Either the primary or secondary input buffer may be used as the source, and either the primary or secondary output buffer may be used as the destination; the buffers used depend on commands executed prior to the A command.

The A command may be used in the following general form:

A{c}{p}{,m}

c is the surround character for primary output buffer.
P is the count of the parameter to be moved
m is the count of characters to be moved

The function parameters c, p and m are mutually independent, and may be used in any combination to achieve the desired result.

p specifies the ordinal number of the parameter to be moved from the input buffer, and resets the input-buffer pointer to the first character of the p'th parameter in the input buffer. If p is not specified, the input-buffer pointer remains pointing to the character after the end of the last character moved, or to the first character of a parameter, if the pointer was prevoiusly set by an S- or Sp-command, or an F- or B-command.

If p is not specified, the parameter is obtained from the currently active input buffer, at the current position of the input-buffer pointer. Leading blanks are deleted from the parameter. The end of the parameter is designated by the first blank which is encountered, unless the entire parameter is enclosed in single quotes, in which case the entire string in the quotes is moved.

When p is used, (where p is a decimal number) the p'th parameter is moved, where parameters are separated by blanks, or single quotes.

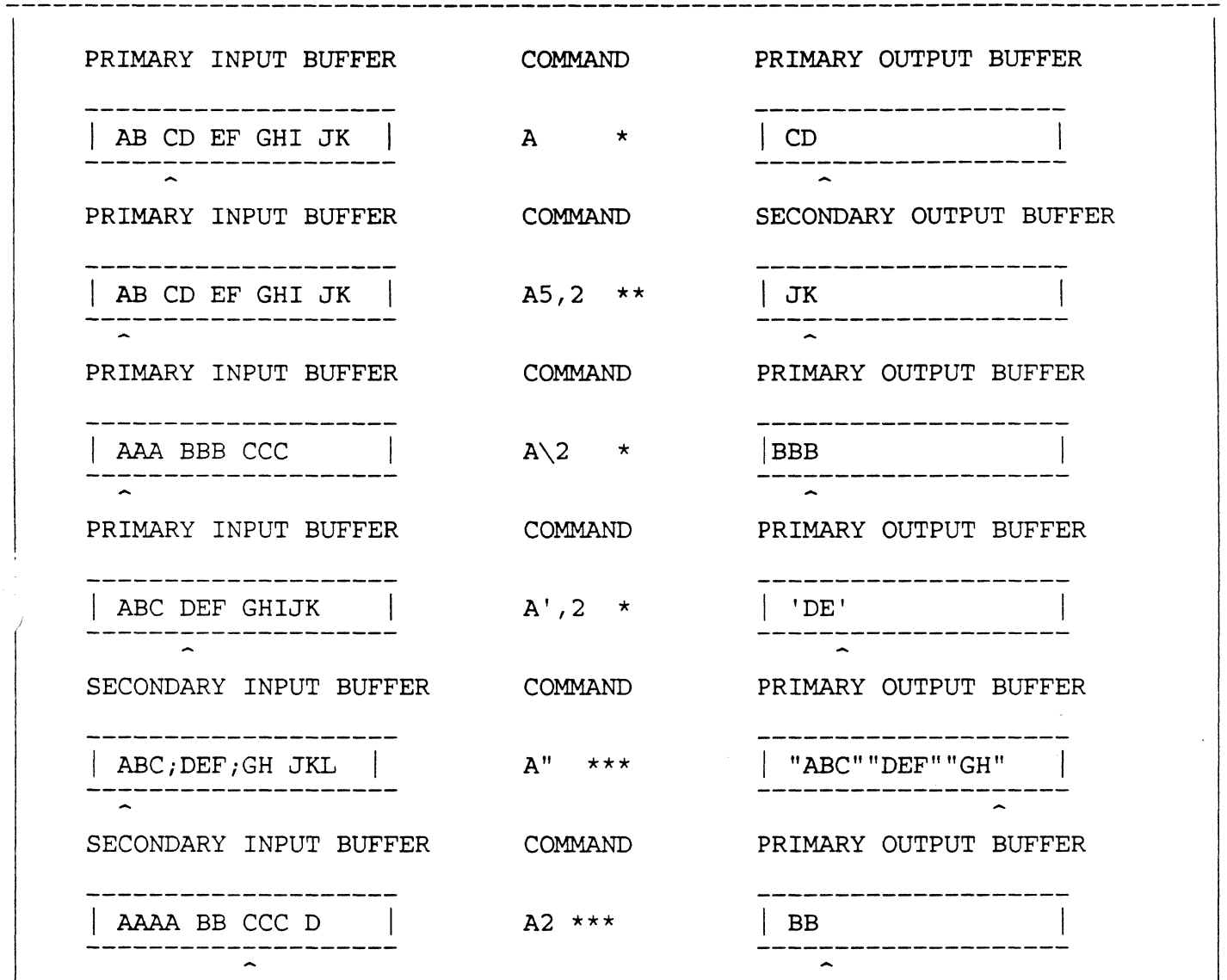
If the PRIMARY output buffer is active (that is, the stack is OFF), the parameter is copied with surrounding BLANKS if c is missing. If the character c is a backslash (\), the parameter is copied without any surrounding blanks. When the form with c is used (where c is any non-numeric character except a left-parenthesis character, the character c surrounds the parameter. This feature is useful for picking up item-ID's and values (which require double quotes) for processing by the ACCESS language Processor. Note that c is INACTIVE when the stack is ON (i.e., parameters are always copied to the stack as they are).

Multiple parameters may be moved to the primary output buffer via a single A command if these parameters are separated by semicolons in the input buffer. The parameters will be moved to the primary output buffer with the semicolons deleted, and surrounded by blanks or the enclosing character c, if c is specified.

After the execution of an A command, the input buffer pointer points to the very next character after the string that was moved. Normally this means the next blank or surround character following the last parameter in the buffer, if any.

If there is no parameter, the A command causes no operation at all.

If the optional m is used, where m is a decimal number, only "m" characters of the parameter are moved to the output buffer. Each example below assumes that the output pointer is at the beginning of the buffer prior to the illustrated operation.



Sample usage of A Command.

The IS command selects the secondary input buffer and accepts input from the terminal. The IP command accepts input from the terminal to the currently active input buffer. And the IT command inputs the next tape label from tape.

The IS command selects the secondary input buffer as the currently active input buffer and inputs data from the terminal into the buffer. The general form of this command is:

IS{r}

If the r specification is used, then that character is a prompt character at the terminal (r may be any character including a blank). The prompt character will remain in effect until a new IS or IP command with a new r specification is executed. If r is omitted, then the TCL prompt is used. Data input by the user in response to the prompt is placed into the secondary input buffer. Subsequently, the data may be moved to an output buffer by using the A command. Any time the IS command is executed, input from the terminal overwrites all previous data in the secondary input buffer.

The IP command inputs data from the terminal into the currently active input buffer. The general form of this command is:

IP{r}

Data input at the terminal in response to an IP command replaces the current parameter (i.e., as pointed to by the input pointer) of the currently active input buffer. If several parameters are input at the terminal, then they will all replace the current parameter in the buffer. If the input pointer is at the end of the data in the input buffer, then the new input data will be appended to the end. The r specification is identical to the r specification for the IS command (see above).

The IT command inputs the tape label from the tape currently attached and copies that label into a cleared currently active input buffer. The general form of the command is:

IT

The IT command will first clear the currently active input buffer and then input the tape label into that buffer. If no tape label exists then the command leaves the active buffer cleared or empty.

Below are the explanations of the commands and options followed by examples and explanations of the input commands.

COMMAND	EXPLANATION
IS	Selects secondary input buffer and inputs data from terminal. Prompt character is a colon (:).
IS=	Selects secondary input buffer and inputs data from terminal. Prompt character is an equal sign (=).
IP?	Replaces current parameter in currently active input buffer with data from terminal. Prompt character is a question mark (?).
IT	Inputs tape label to primary input buffer. If no label then input buffer is cleared.

Sample usage of IS, IP, and IT Commands.

The O command is used to output a specified text string to the terminal. The D command is used to output parameters from either input buffer to the terminal.

FORMAT:

O{text}{+}

The O command causes the text which immediately follows the O to be output to the terminal. If the last character of the text is a plus sign (+), then a carriage return will not be executed at the end of the text output. This feature is useful when using the O command in conjunction with an input command. For example, consider the following commands:

```
OPART-NUMBER+
IS=
```

These commands produce the following output on the terminal:

```
PART-NUMBER=
```

The specified prompt character (=) is displayed adjacent to the output text since the O command ended with a plus sign (+). The user then enters the input data right after the prompt character. For example:

```
PART NUMBER=115020
```

The D command is used to output parameters from either input buffer to the terminal. The D command may be used in the following general form:

D{p}{,n}{+}

If the form Dp is used, then the p'th parameter of the currently active input buffer is displayed on the terminal. If the form D is used, then the current parameter (i.e., as pointed to by the input pointer) of the currently active input buffer is displayed on the terminal. If the form D0 (D followed by the number zero) is used, the complete currently active input buffer is displayed. If the forms Dp,n or D,n are used, then the n characters starting at the p-th. or current parameter (up to the first blank character encountered) are displayed.

A plus sign (+) may be appended to the end of the D command, thus specifying the suppression of a carriage return (as for the O command described above.) The D command does not affect the input pointer.

```
COMMAND                    OUTPUT TO TERMINAL

OTHIS IS AN EXAMPLE        THIS IS AN EXAMPLE [CR]

OTHIS IS AN EXAMPLE+       THIS IS AN EXAMPLE
```

Sample usage of O Command.

```
PRIMARY INPUT BUFFER       COMMAND   OUTPUT TO TERMINAL
--------------------
| AA BBB CC DDD    |       D      *  BBB [CR]
--------------------
     ^
SECONDARY INPUT BUFFER     COMMAND   OUTPUT TO TERMINAL
----------------------
| AA BBB CC DDD      |     D4+    **  DDD
----------------------
     ^
PRIMARY INPUT BUFFER       COMMAND   OUTPUT TO TERMINAL
--------------------
| ABC XYZ 123      |       D,2  ***  XY [CR]
--------------------
      ^

ACTIVE BUFFER PRIOR TO COMMAND EXECUTION

  *primary input buffer
 **secondary input buffer
***primary or secondary input buffer
```

Sample usage of D Command.

5.11 TERMINAL OUTPUT AND CURSOR CONTROL: THE T COMMAND

The T command is used to specify terminal cursor positioning, to output literals, or to output non-keyable character codes. The cursor functions are terminal independent. The special terminal function codes are also availible.

FORMAT:

T {function},{function}, ...

Where {function} is any of the following:

"Text"	Causes the literal text to be output at the current position.
B	Causes a BELL code to be output
C	Causes a Clear Screen code to be output
Inn	Causes the integer character nn to be output.
Xnn	Causes the hex character nn to be output.
(X,Y)	Causes the terminal cursor to position to X,Y. This is controlled by the term type code. The special function codes (-1 thru -10) are also supported.

This command allows the user to create formatted screens in PROCs. The prompting and positioning of formatted screens generally appears cleaner and more acceptable to terminal operators. Note that this command does use the SYSTEM-CURSOR mode and so can be controlled terminal by terminal with the term type code. It is strongly recommended that the user employ the terminal independent control codes -1 thru -10 in place of 'hard coding' these functions for a single terminal type.

The T command may be continued onto multiple lines by ending the preceeding line with a comma. Also comments may be added after the critical command letters. Thus the code to clear the screen 'C' could also be spelled out as 'CLEAR', the code for a bell 'B' could be 'BELL', etc.'. The T command never automaticaly adds a carriage return or line feed.

(-1)	Generates the clear-screen character; clears the screen and positions the cursor at 'home' (upper left corner of the screen).
(-2)	Positions the cursor at 'home' (upper left corner).
(-3)	Clears from cursor positon to the end of the screen.
(-4)	Clears from cursor position to the end of the line.
(-5)	Starts blinking on subsequently printed data.
(-6)	Stops blinking.
(-7)	Initiates 'protect' field. All printed data will be 'protected', that is, cannot be written over.
(-8)	Stops protect field.
(-9)	Backspaces the cursor one character.
(-10)	Moves the cursor up one line.

Explanation of Cursor Function Values.

COMMAND	OUTPUT TO TERMINAL
T C,B,(10,5),"TITLE"	This sequence first clears the screen. It outputs a bell code to the terminal. The cursor is postioned to column 10 row 5. The text "TITLE" is output.
T (0,8),(-4)	This positions the cursor at column 0 row 8. It then clears the entire line assuming that the terminal used supports that function.
T (-5),"twinkle",(-6)	This starts a blinking field, prints the word "twinkle", and ends the blinking field. This assumes the terminal supports blinking.
T CLEAR,"TITLE", (5,5) Comment,"TEXT"	This illustrates the continuation of a command over a line boundary and the insertion of a comment in the line.

Sample usage of the T command.

5.12 SPECIFYING TEXT STRINGS AND CLEARING BUFFERS: THE IH, H, RI, AND RO COMMANDS

The IH and H commands are used to place a specified text string in the currently active input or output buffer, respectively. The RI and RO commands are used to reset the input and output buffers (respectively) to the empty (null) condition.

FORMAT:

IH text

This command causes the text (including any blanks) immediately following the IH to replace the current parameter (as specified by the input pointer) in the currently active input buffer. The input buffer pointer will remain pointing to the beginning of the inserted string.

FORMAT:

H{text}{<}

This command causes the text (including any blanks) which immediately follows the H to be placed in the currently active output buffer at the position pointed to by the output pointer.

When the last parameter of a desired output line has been moved to the secondary output buffer (the stack), a carriage return specification (<) must be placed in the stack. For example, the command HXYZ< would be used to place in the stack the text XYZ followed by a carriage return, while the command H< would place a carriage return (only) in the stack.

FORMAT:

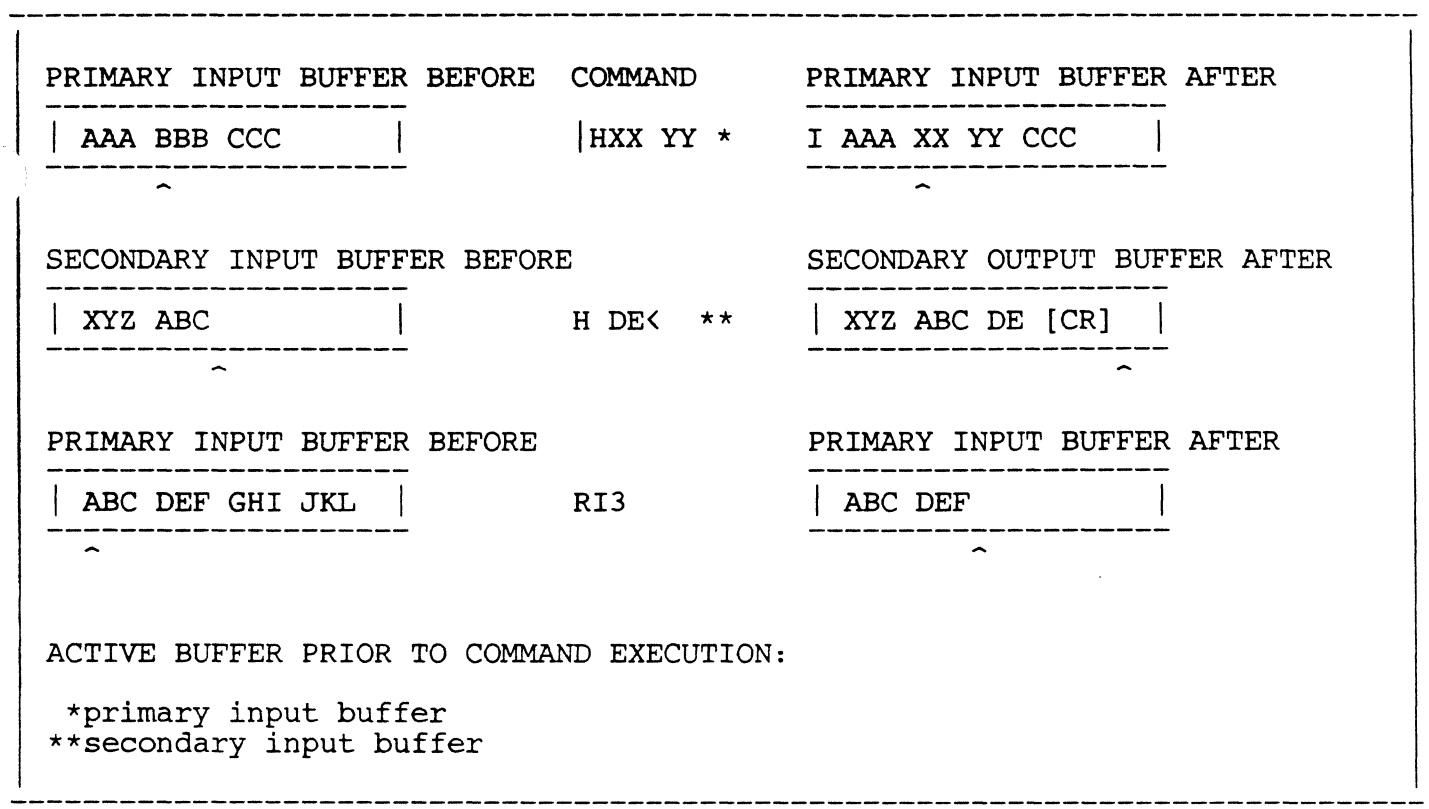
RI{p}

If the form RI is used, then both input buffers are reset to the empty (null) condition. If the form RIp is used, then the primary input buffer from the p'th parameter to the end of the buffer (as well as the entire secondary input buffer) are reset to the empty (null) condition. The RI command always selects the primary input buffer as the active buffer.

FORMAT:

RO

This command resets both output buffers to the empty (null) condition. The RO command always selects the primary output buffer as the active buffer.



Sample usage of IH, H, and RI Commands.

Transfer of control (i.e., branching) may be specified within a PROC via use of the GO. The GO n command provides an unconditional branch capability, while the GO A provides a conditional branch capability based on the value of A.

FORMAT:

```
G{O} n
G{O} A
```

The GO n command causes control to unconditionally transfer to the PROC command which has the numeric label n. For example:

```
G 10
```

This command causes control to transfer to the PROC command which begins with the label 10.

The GO A command causes control to transfer to the PROC command which has the numeric label represented by the parameter A. (Where A equals the parameter being pointed to in the currently active input buffer.) For example:

```
Input Buffer
------------
10 20 30
------------
   ^

GO A
```

This command causes control to transfer to the PROC command which begins with label 20. If label 20 does not exist within the PROC then the GO A command will not be executed. This command is especially useful in PROC's that allow operator job selection. Note that several PROC commands may begin with the same label. If this is the case, the GO command transfers control to the first PROC command with begins with the specified label (scanning from the top).

```
    JOB.SELECT
001 PQ
002 O 1. LIST VENDORS
003 O 2. ENTER NEW VENDORS
004 O
005 O    ENTER JOB # +
006 IP
007 GO A
008 X NO SELECTION
009 1 HSORT VENDORS
010 PX
011 2 HRUN PROGRAMS VENDOR.UPD
012 PX
```

This PROC displays two job alternatives, 1. LIST VENDORS and 2. ENTER NEW VENDORS, and prompts with the message ENTER JOB #.
The entry of a "1" or "2" will transfer PROC control to their respective labels otherwise the PROC will exit to TCL with the message "NO SELECTION".

Sample use of GO A Command.

5.14 CONDITIONAL EXECUTION: THE SIMPLE IF COMMAND

Conditional execution may be specified within a PROC via use of the IF command.

The IF command provides for the conditional execution of a specified PROC command. The IF command takes on three general forms. The simple form is as follows:

```
IF {#}a-cmnd proc-cmnd
```

Where a-cmnd is any legal form of the A command (refer to the topic titled MOVING PARAMETERS: THE A COMMAND) except for the form using the character surround feature (i.e., Ac), and where proc-cmnd is any legal PROC command. If the optional # is not used, the IF command simply tests for the existence of a parameter in the input buffer as specified by the A command. If a parameter exists, the specified PROC command is executed; otherwise, control passes to the next sequential PROC command. For example:

```
IF A2 GO 15
```

This command tests for the existence of a second parameter in the currently active input buffer. If a parameter exists, control passes to the PROC command beginning with label 15; otherwise, control passes to the next sequential PROC command. If the # option is used, the test is reversed. For example:

```
IF #A2 GO 15
```

This command causes control to transfer to the command with label 15 if a second parameter does not exist.

The user should note that when using an A command as a test condition of an IF command, parameters are not moved to an output buffer as would be the case if the A command were used alone. Rather, the A command is used simply to specify which parameter in the input buffer is to be tested. However, the input pointer will be re-positioned as specified by the A command.

NOTE: The following examples assume that the primary input buffer is the currently active input buffer and contains the following parameters:

```
--------------------
| ABC AAA XYZ      |
--------------------
  ^
```

COMMAND	EXPLANATION
IF A GO 27	Control is transferred to the command with label 27.
IF A3 OHELLO	Message HELLO is output to terminal; control then continues with next sequential command.
IF A4 OHELLO	Message is not output; control continues with next sequential command.
IF # A11 GO 2	Control is transferred to the command with label 2.

Sample usage of Simple IF Command.

The relational form of the IF command allows parameters in the input buffers to be tested relationally.

The relational form of the IF command is an extended version of the simple IF form (see topic titled TRANSFERRING CONTROL: THE GO AND SIMPLE IF COMMANDS). The relational form is as follows:

```
If a-cmnd op string proc-cmnd
```

Where a-cmnd and proc-cmnd are as defined for the simple IF form, where op is one of the relational operators listed in Figure B, and where string is a literal string of characters which the parameter is to be compared against. For example:

```
IF A,3 = YES GO 5
```

Here the PROC would transfer control to the command with the label 5 if the current parameter in the currently active input buffer is the character string YES.

To resolve a relational condition, character pairs (one from the selected parameter and one from the literal string) are compared one at a time from leftmost characters to rightmost. If no unequal character pairs are found, the strings are considered to be equal. If an unequal pair of characters are found, the characters are ranked according to their numeric ASCII code equivalents (refer to the LIST OF ASCII CODES in the Appendix to this manual). The character string contributing the higher numeric ASCII code equivalent is considered to be greater than the other string. For example, AAB is considered to be greater than AAAA, and 02 is considered greater than 005.

If the selected parameter and the literal string are not the same length, but the shorter of the two is otherwise identical to the beginning of the longer one, then the longer string is considered greater than the shorter string. For example, the string WXYZ is considered to be greater than the string WXY.

OPERATOR SYMBOL	OPERATION
=	test for equal
#	test for not equal
<	test if parameter less than literal string
>	test if parameter greater than literal string
[	test if parameter less then or equal to literal string
]	test if parameter greater than or equal to literal string

Relational Operators.

NOTE: The following examples assume that the primary input buffer is the currently active input buffer and contains the following parameters:

```
--------------------
| ABC AAA XYZ      |
--------------------
  ^
```

COMMAND	EXPLANATION
IF A = ABC GO 3	Control is transferred to the command with label 3.
IF A3 > XYX HTEST	The text string TEST is placed in the currently active output buffer; control then continues with next sequential command.
IF A2 > XYX HTEST	Text string TEST is not placed in output buffer; control continues with next sequential command

Sample Usage of Relational IF Command.

5.16 PATTERN TESTING: THE PATTERN MATCHING IF COMMAND

The pattern matching form of the IF command allows parameters in the input buffers to be tested for a specific pattern match.

The pattern matching form of the IF command is an extended version of the simple IF form (see topic titled TRANSFERRING CONTROL: THE GO AND SIMPLE IF COMMANDS). The pattern matching form is as follows:

```
IF a-cmnd op (pattern) proc-cmnd
```

Where a-cmnd and proc-cmnd are as defined for the simple IF form, where op is one of the relational operators described for the relational IF form, and where pattern is a pre-defined format string enclosed in parentheses. A pattern is used to test a parameter for a specified combination of numeric characters, alpha characters, alpha-numeric characters, or literals. The pattern specification in an IF statement consists of any combination of the following:

- An integer number followed by the letter N (which tests for that number of numeric characters).
- An integer number followed by the letter A (which tests for that number of alpha characters).
- An integer number followed by the letter X (which tests for that number of alpha-numeric characters).
- A literal string (which tests for that literal string of characters).

As an example, consider the following command:

```
If A = (3NABC) G 3
```

This command causes a transfer of control to the command with label 3 when the current parameter of the currently active input buffer consists of three numerals followed by the characters ABC (e.g., 123ABC).

If the integer number used in the pattern is 0, the test is true only if all the characters in the parameter conform to character type. The following command, for example, outputs the message OK if the characters of the current parameter are all alpha characters:

```
IF A = (0A) OOK
```

Note that for any of the three IF command forms, the PROC statement which is conditionally executed may in turn be another IF command (i.e., IF commands may be nested). The following command, for example, transfers control to label 99 if the current parameter consists of two numerals in the range 10 through 19 (inclusive):

```
IF A = (2N) IF A ] 10 IF A [ 19 GO 99
```

The user may wish to visualize nested IF commands as though implied AND operators were placed between them.

NOTE: The following examples assume that the primary input buffer is the currently active input buffer and contains the following parameters:

```
-------------------------------
| ABC 10/09/77 XYZ B123C 33   |
-------------------------------
  ^
```

COMMAND	EXPLANATION
IF A = (3A) G 7	Control is transferred to the command with label 7.
IF A2 = (2N/2N/2N) G 5	Control is transferred to the command with label 5
IF A4 = (0N) G 9	Control continues with next sequential command.
IF A5 = (0N) GO 2	Control is transferred to the command with label 2.
IF A4 = (1A3NC) OGOOD	The message GOOD is output to the terminal; control continues with next sequential command.
IF A1 = (3X) IF A1 > ABB G 9	Control is transferred to the command with label 9.

Sample Usage of Pattern Matching IF Command.

5.17 FURTHER FORMS OF THE IF COMMAND: THE IF E and IF S COMMANDS and SELECT LIST AND PROC INTERACTION

The IF E form of the IF command may be used to test for errors generated by a preceding PROC-generated statement. The IF S form of the IF command may be used to test whether a LIST, as generated by a SELECT, SSELECT, QSELECT or GET-LIST statement, is in effect.

FORMAT:

IF {#}E {op string} proc-cmnd

Where "op string" and "proc-cmnd" are as defined previously.

This command allows PROCs to test for system generated errors (as specified in the ERRMSG file). The E command is valid only after a P type command, that is, when a PROC-generated statement has completed execution, and control is returned to the PROC. The E command uses the secondary input buffer, and therefore is valid only until an "IS" command is executed.

The errors tested for may be unspecified (i.e. Any error) or they may be specified by the error number. The relational operators "=",">","<", "[","]" may also be used to test for errors in specified ranges. Thus the error command may be used in two ways. An example of the first would be :

```
015 IF E X ENCOUNTERED AN ERROR AT LINE 15
```

whereby control will transfer to TCL and the text "ENCOUNTERED AN ERROR AT LINE 15" will be printed if any error were encountered.

An example of a statement that tests for an error range is :

```
015 IF E > 91 IF E < 99 X TAPE ERROR!
```

in which case control will transfer to TCL and the text will be printed if an error in the range 92-98 had been encountered.

There are certain TCL statements that select lists of item-ids or values, such as SELECT, SSELECT, QSELECT and GET-LIST. Refer to the appropriate areas of the documentation for details regarding these statements. There is an important interaction between these statements and a PROC. A selected list must be used by the TCL statement immediately following it, or else it will be lost. If the select-type statement has been executed by a PROC, the TCL statement that uses it is normally placed in the STACK prior to execution of the select statement. This second TCL statement will automatically execute after the select is complete; the PROC will not gain control in between! If there is a null line in the STACK, the PROC will then regain control. The PROC may then test if the select statement executed correctly.

The general form of the IF S command is:

IF {#}S proc-cmnd

where proc-cmnd is any PROC command.

This command will test for the presence of a selected-list; the selected list will be present only if a select-type TCL statement has already been executed at the time that the IF S command is encountered.

If the select statement has generated an error, such as "NO ITEMS PRESENT" or "ITEM NOT ON FILE", the select list will not exist, and the IF S may be used to check on this condition.

```
    TEST1                                  TEST2
001 PQ                                 001 PQ
002 HGET-LIST                          002 HGET-LIST
003 OENTER LIST-NAME+                  003 ENTER LIST-NAME+
004 IP?                                004 IP?
005 A                                  005 A
006 STON                               006 STON
007 H<                                 007 HLIST INVENTORY LPTR
008 P                                  008 P
009 IF #S XILLEGAL LIST-NAME!          009 next statement
010 HLIST INVENTORY LPTR
011 P
012 next statement
```

The PROCs TEST1 and TEST2 will operate identically if the GET-LIST statement executes without an error (that is, if the list exists on file). However, TEST2 will continue with PROC execution even if the list is not on file, since there cannot be an IF S test after the stacked LIST statement executes. TEST1, on the other hand, has a null line in the stack when the GET-LIST executes; therefore, control is returned to the PROC, which can test to see if it executed properly. If the list is not on file, the PROC will terminate on line 10.

Sample usage of the IF S command, and of PROC-SELECT interface.

5.18 ADDITIONAL FEATURES: THE PLUS (+), MINUS (-), U AND C COMMANDS

The Plus and Minus commands are used to add or subtract (respectively) a specified decimal number to/from the current parameter of the currently active input buffer. An exit to a user-defined subroutine may be accomplished via the U command. The C command is used to place comments within the body of the PROC.

FORMAT:

+n

This command causes the decimal number n to be added to the current parameter (as pointed to by the input pointer) of the currently active input buffer. The current parameter must be numeric.

FORMAT:

-n

This command causes the decimal number n to be subtracted from the current parameter (as pointed to by the input pointer) of the currently active input buffer. The current parameter must be numeric.

The Plus or Minus commands will have no effect if the input pointer is currently at the end of the buffer. Also, the user must take care that the updated value of the parameter is the same length as the original parameter, since no automatic check for this is made.

FORMAT:

Umode-id

The U command is used to provide an exit to a user-defined subroutine. The format for this command is identical to the P command using the mode-id option; however, the U command is meant to be used for a simple subroutine call. Upon return from the subroutine, control is passed to the command immediately following the U command. TCL is not involved in the execution of a subroutine via the U command. (For further information, see the Pick Assembly Language Reference Manual).

WARNING: Do not use the U command unless you fully understand its action at the system assembly level!

The C command is used to place comments within the body of the PROC.

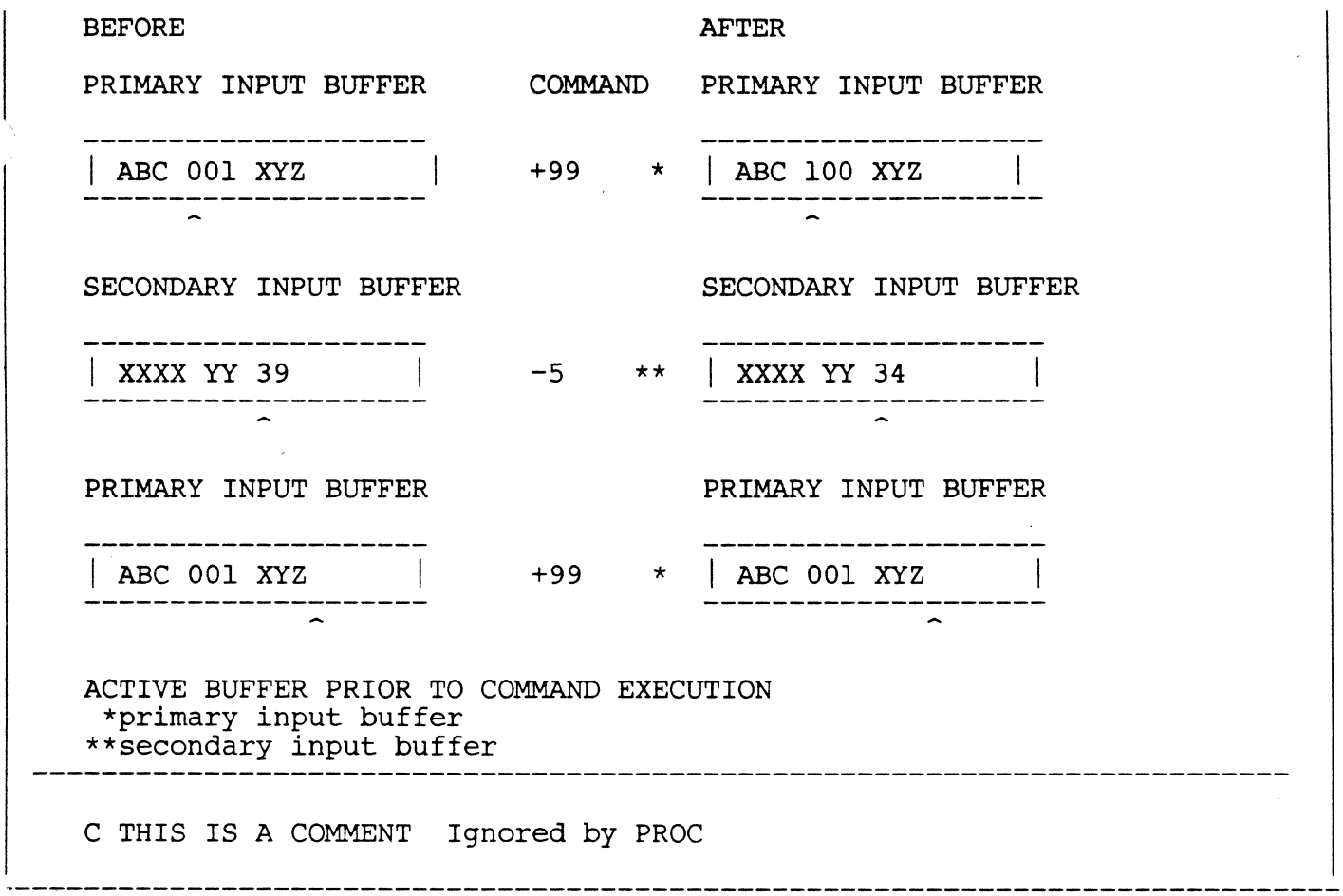
FORMAT:

C{text}

All the text following the C will be ignored by the PROC processor. For example:

013 C THIS IS A COMMENT

The C command may be used freely throught the PROC for purposes of clarity and documentation; however, note that making a PROC excessively long will slow its execution!



Sample Usage of Plus (+), Minus (-) and C Commands.

The P command causes the PROC to execute a TCL command. The PH command is similar to the P but causes terminal output to be suppressed. The PX command acts like the P, but turns off any further interaction with the PROC. The PP and PW commands are identical to the P command, except that the content of both output buffers are displayed at the terminal prior to execution. The X command is used to exit from the PROC.

FORMAT:

P{c} where c = H, P, W or X

The P command causes the PROC to execute by submitting the content of the primary output buffer to TCL for processing; the contents of the stack (if any) are used to feed interactive processors such as PICK/BASIC or EDITOR. After execution via TCL,the PROC regains control at the statement immediately following the P command.

A carriage-return is placed in the STACK, if the stack has any data in it, and a carriage-return specification is not present as the last character in the last line.

The PX command acts just as the P does, with the exception that control is NOT returned to the PROC after the TCL statement has been executed.

The PP command causes execution of the contents of the buffer just as the P command, except that the content of both output buffers are displayed on the terminal.

The PW command acts in the same way as the PP command, except that after the data are displayed, terminal input is then requested via a question mark (?) prompt character. If the user enters an S, the current PROC-generated command is skipped, and PROC execution continues at the command following the PW. If an X is entered, PROC execution is aborted and an exit is taken to TCL. Any other character will cause PROC action to continue. The PW command is normally used as a debugging tool and may be replaced by a P command once the user has determined that the PROC is functioning properly.

The PH command executes the buffer but suppressed any output from the executed process.

FORMAT:

X{text}

The X command is used to exit from the PROC. Normally, PROC control is terminated with execution of the final PROC statement, in which case an X command is not needed. However, the X command may be used at intermediate points in the PROC coding to cause termination of the PROC. Any text following the X will be output as a message upon termination of the PROC. For example:

X***EXIT TO TCL***

If the PROC was called as a subroutine the X command will cause a return to the calling PROC.

COMMAND	EXPLANATION
X	PROC is terminated.
XHURRY BACK	PROC is terminated and the massage HURRY BACK is displayed on the terminal.
XHURRY BACK+	As above; the message is printed without a carriage-return/line-feed appended.
X	If the PROC was called as a subroutine then X will return control to the calling PROC and continue at thenext command.

Sample Usage of P, PP, X, and U Commands.

5.20 LINKING TO OTHER PROCS: THE LINK COMMAND

| One PROC can invoke another PROC via use of the Link command. |

A Link command in one PROC causes control to transfer to the first command of another PROC, which may reside in any dictionary or data file. This allows the storage of PROC's (except for the LOGON PROC) outside of the M/DICT. Also, large PROC's can be broken into smaller PROC's to minimize processing time.

FORMAT:

({DICT} file-name {item-id}) {n}

Where the file-name specifies the file and the item-id specifies the name of the PROC invoked. If the item-id is omitted, the current parameter (as specified by the input pointer) of the currently active input buffer is retrieved and used as the item-id. The optional DICT specifies the dictionary portion of the file.

The first line of the linked-to PROC is skipped, since it is assumed that this line contains the PQ code.

If the optional 'n' is used, control is transferred to the line whose label is 'n'.

As an example of the Link command, consider the situation where a PROC named EXECUTE is used to execute any one of a series of PROC's in a file named PROC-FILE. The specific PROC executed is specified by a single-character alaphabetic code input by the user. This sample PROC is shown in the examples below. If, for example, the user's response to the IS command of line 3 is the character D, then line 4 of the PROC (which contains a Link command as part of the IF command) transfers control to the PROC stored in item 'D' of FILE PROC-FILE.

Consider next the situation where the PROC named LISTU previously was present in each user's M/DICT. Assume that LISTU was then moved to the dictionary section of a file named PROCLIB, and a new LISTU PROC (as shown in the examples) was then placed in each user's M/DICT. The LISTU PROC which was moved to the PROCLIB file will now be invoked by the Link command in the PROC shown in the examples, and thus the LISTU PROC need not be duplicated in each user's M/DICT.

Note that the PROC buffers remain unchanged when a linkage occurs. Also, the first line of the linked-to item is always skipped, since it is assumed that this line contains the PQ code.

```
Item EXECUTE

001 PQ
002 OPLEASE INPUT CODE+
003 IN?
004 IF A = (1A)  (PROC-FILE )
005 XILLEGAL RESPONSE
```

Sample Usage of Link Command (See Text).

```
Item 'LISTU' in M/DICT

001 PQ
002 (DICT PROCLIB LISTU)
```

Sample Usage of Link Command (See Text).

One PROC can call another as a subroutine, or a local subroutine call can be invoked using the call command.

FORMAT:

[] n

This command will store the location of the next PROC command in the PROC subroutine stack, and transfer control to the command whose label is n. Execution of PROC commands continues from that point (including P, PP and PW commands), until an X command is executed, which will return control to the PROC command following the call command.

FORMAT:

[{DICT} file-name {item-id}] {n}

Where "DICT", "file-name" and "item-id" are identical to the LINK command decribed in the last section. If item-id is not spectified, the name of the called subroutine is taken from the current parameter (as specified by the input pointer) of the currently active input buffer.

The optional n indicates that subroutine execution is to begin at label n, rather than at the second line of the subroutine PROC.

As with external subroutine calls, an X command will return control to the calling PROC.

In both forms of the subroutine call, none of the input or ouput buffers are affected by the call itself.

```
                    Local Subroutine Call
001 PQ
002 [] 3
003 OFIRST
004 3 OSECOND
005 X+                  NOTE: '+' sign supresses carriage
                              return after X returns.

                      Output on terminal
SECOND
FIRST
SECOND

                   External Subroutine Call
001 PQ
002 [MD LISTU]
003 ODONE WITH LISTU

                      Output to terminal

CH# PCBF NAME..... TIME... DATE.... LOCATION.........

 00 0200 SP           08:00AM 01/01/78 Channel 0
 02 0240 CM           09:10AM 01/01/78 Channel 2
 03 0260 LC           07:30AM 01/01/78 Channel 3
 04 0280 JP           10:14AM 01/01/78 Channel 4
*06 02C0 SAL          08:35AM 01/01/78 Channel 6
 10 0340 JET          09:00AM 01/01/78 Channel 10
 DONE WITH LISTU
```

Sample Usage of Local and External Subroutine Calls.

5.21.1 SAMPLE PROCS: FILE UPDATE VIA EDITOR

This topic presents a sample PROC which changes a specified attribute value via the EDITOR.

The first example shows a sample EDITOR operation which changes attribute 3 of item 11115 of file ACCOUNT to the value ABC. The second example shows a PROC named CHANGE which will perform the exact same operation. Note that the PROC has been written in such a manner that it updates any specified attribute in any specified item in any specified file. The format used to invoke this PROC is as follows:

CHANGE file item attribute-no new-value

If, for examplethe user wishes to perform the same operation shown in the first example, the PROC must be invoked as follows:

CHANGE ACCOUNT 11115 3 ABC [CR]

The user should note that the normal messages output by the EDITOR (e.g., TOP, '11115' FILED, etc.) are output when the PROC in the second example is executed. These messages may be suppressed, however, by preceding each EDITOR command by a period (.); for further information regarding these features, refer to the EDITOR Reference Manual.

```
>EDIT ACCOUNT 11115 [CR]
TOP
.G3 [CR]
003 100 AVOCADO
.R [CR]
003 ABC
'11115' FILED
```

Sample EDITOR Operation.

Item CHANGE IN M/DICT

```
001 PQ
002 HEDIT
003 A2
004 A3
005 STON
006 HG
007 A4
008 H<
009 HR<
010 A5
011 H<
012 HFI<
013 P
```

Generalized PROC Stored As Item 'CHANGE' Which
Will Perform Identical Operation.

5.21.2 USING SSELECT AND COPY VERBS

This topic presents a sample PROC which uses the SSELECT and COPY verbs.

The first example shows a sample operation at the TCL level using the SSELECT verb and then the COPY verb. This identical operation is performed by the PROC named TEST shown in the next example. Upon execution of the TEST PROC, the output buffers contain the data shown in the last example. Note that the SSELECT sentence is contained in the primary output buffer, while the secondary output buffer contains both input elements of the copy operation, each terminated by a carriage return.

```
>SSELECT INVENTORY WITH QOH > "900" BY-DSND QOH [CR]
19 ITEMS SELECTED
>COPY INVENTORY [CR]
TO: (HOLD-FILE) [CR]
19 ITEMS COPIED
```

SSELECT and COPY Operation at TCL Level.

```
Item 'TEST' in M/DICT

001 PQ
002 HSSELECT INVENTORY WITH QOH > "900" BY-DSND QOH
003 STON
004 HCOPY INVENTORY<
005 H(HOLD-FILE)<
006 P
```

PROC Stored as Item 'TEST' Which Performs Identical SSELECT and COPY Operations.

```
PRIMARY OUTPUT BUFFER

| SSELECT INVENTORY WITH QOH > "900" BY-DSND QOH [CR]   |

SECONDARY OUTPUT BUFFER

| COPY INVENTORY [CR] (HOLD-FILE) [CR]                  |
```

Output Buffers Upon Execution of TEST PROC.

5.21.3 USING VARIABLE TESTING, GO AND D COMMANDS

This topic presents a sample PROC which uses variable testing, GO and D commands.

The example shows a sample tape positioning PROC. It differs from previous examples in that it uses the arithmetic command. It has practical value in that the user does not have to enter T-FWD at the TCL level for every file that is positioned over.

Note that the PROC may be executed by entering "T-SPACE" or "T-SPACE n" at the TCL level; if no parameter is entered, the PROC will request one.

The input buffer contains the following data:

```
Parameter#      1         2         3         4
             T-SPACE      n         xx
```

where "n" is the number of files to be spaced over, and xx is the count of such files, initialized to "00" at line 6.

The PROC will attach the tape unit (T-ATT); check for errors 95 and 93, terminating execution if either error occurs. Then it will execute a T-RDLBL to read the tape label and print it on the terminal; if error 94 (EOF) occurs on this statement, the end of tape data has been reached; the message on line 31 will be printed, along with the file-count from parameter 3.

T-RDLBL executes sucessfully, the tape file is spaced over by executing T-FWD (the print is turned off and on around this command by the commands P (I) and P (L), to inihibit spurious messages).

This is repeated until parameter 2 goes to 0; note the multiple tests required to test for 0, 00 or 000 on lines 24-26, since the - command doesn't change the parameter size.

```
    T-SPACE
001 PQ
002 4 IF #A2 GO 3
003 IF A2 = (0N) G 7
004 IH000
005 7 S3
006 IH00
007 HT-ATT
008 P
009 IF E = 95 X
010 IF E = 93 X
011 2 HT-RDLBL
012 P
013 IF E = 94 GO 9
014 HP (I)
015 P
016 HT-FWD
017 P
018 HP (L)
019 P
020 S3
021 +1
022 S2
023 -1
024 IF A = 0 X
025 IF A = 00 X
026 IF A = 000 X
027 GO 2
028 3 ONO. OF FILES+
029 IP?
030 GO 4
031 9 OEND OF RECORDED DATA - (+
023 D3+
033 X FILES)
```

Sample Tape Positioning PROC.

SECTION 6

ACCESS

Chapter 6

ACCESS

THE PICK SYSTEM

USER MANUAL

The user forms ACCESS sentences which specify the desired data retrieval functions. The ACCESS retrieval language is limited natural English; formats for sentences are simple yet very general. The ACCESS processors, together with the use of dictionaries, permit inputs to be stated directly in the technical terminology natural to each application area.

ACCESS accepts any number of variable length words and permits a general freedom of word order and syntax. An ACCESS sentence is entered at the TCL level. The sentence then directs the appropriate ACCESS processor to perform the specified data retrieval function.

The verb must be the first word in the ACCESS sentence, while the other words may generally be in any order. ACCESS verbs are action-oriented words which invoke specific ACCESS processors. The file-name specification permits the access of either the data section or the dictionary section of a file. A verb and a file-name are required; all other elements are optional. Thus, the minimum ACCESS sentence consists of a verb followed by a file-name.

The selection criteria determine which items in the file will be operated upon. If nothing is specified, then all items will be used. One or more direct references may be made by specifying the item-id in single quotes. A conditional retrieval may be specified by using a WITH clause. All items in the file will be examined, but only those meeting the specified criteria will be accepted. The WITH clause may be a simple or complex combination of attribute names, relational operators (=, >, LT, AFTER, etc.), logical operators (AND, OR), and explicit data values ("100", "12/2/76", "RESISTOR", etc.).

The attribute list specifies those attributes desired for output. The attribute list may be explicity stated using attribute names found in the file dictionary. If none are specified in sentence, the implicit attribute synonym list in the file dictionary will be used to specify the display fields.

The miscellaneous modifiers may be used to modify the effect of the verb, or to alter the display format.

Each ACCESS sentence must begin with one (and only one) ACCESS verb. ACCESS verbs are action-oriented words which invoke specific ACCESS processors. Some of the major ACCESS verbs are briefly discussed below.

LIST and SORT

The LIST and SORT verbs are used to generate formatted output. LIST simply lists the selected output, while SORT orders the output in some specified sorted order. Generated output will be formatted into a columnar output if possible, taking into account the maximum defined size of the specified attributes and their associated names, along with the width of the terminal page. If more attributes have been specified than will fit across the page, a non-columnar output will be generated with the attribute names down the side and the associated attribute values to the right. LIST and SORT will automatically format multi-valued attributes and sub-values. They provide sub-totaling via the BREAK-ON and TOTAL modifiers, as well as other format controls. Sample use of the LIST verb with non-columnar output is shown in the first example. SORT can handle any number of ascending or descending sort keys.

COUNT

The COUNT verb counts the number of items meeting the conditions specified. The output generated by this verb is simply the number of items counted. The second example illustrates the use of the COUNT verb.

SUM and STAT

The SUM and STAT verbs provide a facility for summing one specified attribute The STAT verb additionally provides a count and average for the specified attribute. The output generated by these verbs are the derived statistics. The third example illustrates the use of the SUM verb.

SELECT and SSELECT

The SELECT verb provides a facility to select a set of items. These selected items are then available one at a time to certain Pick processors. The output from the SELECT verb is a message signaling the number of items extracted or selected. The SSELECT verb combines the SORT capability with the SELECT capability.

T-DUMP, I-DUMP, ISTAT, HASH-TEST, and CHECK-SUM

The T-DUMP and I-DUMP verbs allow the user to selectively dump his dictionaries and data files to the magnetic tape or to the terminal, respectively. The ISTAT and HASH-TEST verbs provide file hashing histograms. The CHECK-SUM verb is used to determine if data in a file has been changed.


```
* >LIST ACCOUNT "23080" "23090" NAME ADDRESS START-DATE CURR-BALNC [CR]


PAGE 1                                          11:29:58    16 JAN 1984

ACCOUNT  :  23080
NAME  J W YOUNG
ADDRESS  207 COVE STREET
START-DATE  27 MAR 1970
CURR-BALNC  $    89.32

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

2 ITEMS LISTED
```

Sample ACCESS Inquiry Using LIST Verb.
(Non-Columnar Output)

```
*  >COUNT ACCOUNT GE '15' WITH CURR-BALNC AND WITH BILL-RATE "30"  [CR]

2 ITEMS COUNTED.

*  >COUNT ACCOUNT WITH NO SEWER-ASMT  [CR]

57 ITEMS COUNTED

*  >COUNT TEST  [CR]

10 ITEMS COUNTED.

*  >COUNT DICT INVENTORY WITH D/CODE "A"  [CR]

55 ITEMS COUNTED.
```

Sample ACCESS Inquiries Using COUNT Verb.

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

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

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

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

Sample ACCESS Inquiries Using SUM Verb.

6.3 ACCESS INPUT SENTENCES

ACCESS is a report-generating language which enables the user to make various types of listings and queries quickly and easily. It is also used to select items from a file for use by other processors.

To form an ACCESS statement, the user types in a command sentence at the TCL level. ACCESS allows the user to select or list the information contained in all or some of the items in a particular file.

FORMAT:

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

ARGUMENTS:

A verb and a file-name are required; all other elements are optional. The verb specifies generally what type of processing will be performed on the file. The file-name must be of one of the following standard forms:

FORM OF FILE-NAME	EXAMPLE
file-name	BP
dictname,dataname	PROJ,GREEN-ACRES
DICT file-name	DICT M/DICT

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

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

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

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

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

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

Various modifiers and/or options control listing parameters such as double-spacing, how to handle totals, control breaks, supression of item-ids, sort keys, headings or default messages.

NOTES:

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

EXAMPLES:

```
> LIST INVENTORY  [CR]          .....a minimum ACCESS sentence

> LIST DICT MD WITH D/CODE = "PQ" (H)  [CR]

> SORT ACC BY NAME NAME TOTAL CHARGE-UNITS  [CR]

> SORT STAT-FILE WITH REEL# = "1" BY B/M/S (P)  [CR]

> T-DUMP CUSTOMER-MASTER WITH BEGIN-DATE BEFORE "1/1/70" [CR]

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

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

1. ACCESS input sentences are entered either at the TCL level, (when the system prompts with the sign ">"), or into the PROC primary output buffer.

2. The first word of any ACCESS input sentence must be an ACCESS verb defined in the user's Master Dictionary (MD or M/DICT.)

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

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

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

6. Any number of modifiers and relational operators may be used which have been pre-defined in the MD.

7. Verbs, file-names, attribute names, modifiers, and relational operators are delimited (separated) in an ACCESS sentence by blanks, by quotes (" or '), or by backslashes (\).

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

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

6.5 ACCESS DICTIONARIES AND ATTRIBUTE-DEFINITION ITEMS

ACCESS uses dictionary items to define the data-structure in the file. Dictionary items are the user-defined vocabulary used in the ACCESS statement.

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

Each dictionary item serves to:

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

EXAMPLE:

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

"ACCOUNT"                  is the filename;
"CURR-BALNC" and "NAME"    are items in the ACCOUNT dictionary
                           and therefore "attribute names";
"WITH"                     is a modifier and is in the MD;
"LPTR"                     is a modifier and is in the MD .
```

The dictionary items that define the data format for ACCESS processing are called "attribute-definition" items. Line 1 of an attribute-definition item has the Code "A" (therefore these dictionary items are called "A"-items). Other lines contain data as described below:

LINE	DATA	BRIEF DESCRIPTION
1	"A"	Defines attribute-definition item.
2	AMC	Numeric value defining the location of the data defined by this item, (attribute mark count). May contain 0 if referencing the item-id, or a dummy value if the data referenced is computed or generated but not actually stored. It is used to identify controling and dependant attributes. In addition, an amc of 9999 is used to access the SIZE or count field of the item; an amc of 9998 is used to access the current item counter (item sequence number).
3	TAG	Textual data used as tag on heading in LIST or SORT statements. If null, the item-id is used as the tag. May contain blanks for formatting purposes. The reserved character "\" is used to specify a null tag. Multiple line tags for COLUMNAR listings only may be specified by storing multiple values (separated by a value-mark, control-]) in this field.
4	STRUCT	Defines the "controlling-dependent" relationship.
5	Not used.	
6	Not used.	
7	CONV	Contains the conversion specification(s) which is (are) used to convert from the processing format to the external (displayed) format. Multiple specifications are separated by value-marks (control-]).
8	CORR	Contains the correlative specification(s) which is(are) used to convert from the internal format to the processing format. Multiple specifications are separated by value-marks (control-]).
9	TYPE	Defines the justification (left or right). An entry is mandatory, and must be an "L", "T", "U", or an "R". This code is used both in formatting the output, and in determining the sort sequence when sorting data. An "R" is used to specify a right-justified numeric sort (even for alphanumeric fields); an "L" will always sort left-to-right, and will left-justify, folding at the end of the field; a "T" will left-justify, and if the value exceeds the specified maximum length will fold at blanks. A "U" code causes left-justification without folding.
10	MAX	Defines the maximum length of values for the attribute; an entry is a decimal number, or an "L" or "R" followed by a number and is mandatory. A value of zero may be used to suppress the listing of a control-break field on detail lines.

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

```
                    PART#                    ONE
               001 A                    001 A
               002 1                    002 1
               003 PART#                003 ONE
               004                      004
               005                      005
               006                      006
               007                      007
               008                      008
               009 R                    009 L
               010 10                   010 10
```

Sample A-items in the INVENTORY file.

Sample ACCESS statements:

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

2. >SORT INVENTORY BY ONE ONE [CR]

Sample PART#'S

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

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

Example of different sort sequences and output justification using synonym A-items.

6.6 ACCESS AND THE FILE STRUCTURE

ACCESS is designed to take advantage of the file structure available on this machine. Usually, the data is in the data section of the file and the data definition items are in the dictionary of the file.

ACCESS files are made up of elements found in files, and values related to the actual data to be retrieved. The verb definition, file definitions, modifiers and relational operators must be in the master dictionary. Attribute definition items are normally found in the dictionary of the data file to which they relate.

6.6.1 THE USING CONNECTIVE.

The USING connective allows the specification of the file to be used as the dictionary in the ACCESS sentance in place of the standard dictionary.

FORMAT:

USING DICT FILENAME

USING FILENAME

The first references the dictionary of the file FILENAME; the second references the data-level file FILENAME. Note that the data-level file may be of the form FILENAME,SUBFILENAME. In these cases all data definition items will be taken from the file referenced by the USING connective, except those data definition items which default to the master dictionary, as below.

Only one USING connective is allowed in an ACCESS sentence. The USING connective must be immediately followed by either DICT FILENAME or FILENAME. The source of the data processed remains specified by the conventional file name element in the sentence.

LIST WORKFILE USING DICT TESTDICT	The data source will be WORKFILE. The data definition items will be retrieved from DICT TESTDICT. The default data definition items will be used.
LIST DICT WORKFILE USING DICT WORKFILE	The data and data definition items have the same source.

Examples of the USING connective.

6.6.2 MASTER DICTIONARY DEFAULT MASTER DICTIONARY DEFAULT

If the data definition item is not found in the dictionary specified for the file, then the ACCESS compiler will search the master dictionary for the data definition item, and will include it if found.

6.6.3 SEQUENCE OF RETRIEVAL (items from files)

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

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

6.6.4 ITEM-ID DEFINITIONS WITH Q-POINTERS

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

The justification is of importance to both the sort and output processors; the field length is of importance to the output processor, especially if the item-ids are significantly longer than the file name or its nominal field length, and especially in the columnar processor. Note that item-ids do not fold, unlike other data processed by the columnar processor. All of the characteristics of item-id handling may be got around by using a data definition item which references data attribute 0 with ID-SUPP. If a Q-pointer is used to reference a file, the contents of attributes 7, 9, and 10 in the Q-pointer definition take precedence over the those attributes in the D-pointer if they exist in the Q-pointer. Those that do not exist in the Q-pointer will be retrieved from the D-pointer. In the case that they do not exist in either, attribute 7 will be defined as null, attribute 9 will become L and attribute 10 will become 9. These defaults will be taken for all data definition items processed by the ACCESS compiler.

This allows the creation of multiple Q-pointers which treat the item-id field in different ways.

6.6.5 DELIMITERS AND ITEM-ID STRUCTURES

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

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

```
LIST MD CAT DOG RAT
```

The error message

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

Will be returned.

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

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

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

EXAMPLE:

If DOG RAT is an attribute definition item in MD, then

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

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

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

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

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

Similarly, the form

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

Examples of infrequent but legal item-ids.

ACCESS verbs are action oriented words which evoke specific ACCESS processors. The common ACCESS verbs are briefly discussed below.

Each ACCESS sentence must begin with only one ACCESS verb. The verbs specify generally what is to be done to the data in the file.

LIST and SORT: LIST-LABEL and SORT-LABEL

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

COUNT

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

SUM and STAT

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

SELECT and SSELECT

The SELECT verb provides a facility to select a set of item-ids or values using the item-list and the selection-criteria. SELECT generates a list of item-ids or values; SSELECT generates a sorted list. The list is then available to other ACCESS or TCL-II processors. The very next ACCESS or TCL-II verb executed will have access to this list, so the set of item-ids or values processed by the next verb will be those selected by the SELECT or SSELECT verb.

SAVE-LIST, GET-LIST, and DELETE-LIST

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

T-DUMP and T-LOAD

The T-DUMP verb allows the user to selectively dump his dictionaries and data files to the magnetic tape. The T-LOAD verb allows the selective re-loading of data or dictionary items that have been previously dumped to tape using T-DUMP. Data in a T-DUMP-ed tape may also be listed by using the LIST, LIST-ITEM or LIST-LABEL verbs with the TAPE modifier.

ISTAT and HASH-TEST

HASH-TEST and ISTAT provide useful file management information. These verbs give a file hashing histogram, and file utilization statistics; the ISTAT verb is used to provide information for an existing file, and the HASH-TEST verb to provide information about a file using a test modulo, typically prior to the re-allocation the extents of the file.

LIST-ITEM and SORT-ITEM

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

COUNT	SELECT	SUM
HASH-TEST	SORT	T-DUMP
ISTAT	SORT-LABEL	T-LOAD
LIST	SSELECT	LIST-ITEM
LIST-LABEL	STAT	SORT-ITEM
SAVE-LIST	GET-LIST	DELETE-LIST

ACCESS Verbs

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

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

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

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

>COUNT INV WITH PRICE ".30"  [CR]

>SUM FILE4 QUAN  [CR]

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

Sample ACCESS Sentences

6.8 RELATIONAL OPERATORS AND LOGICAL CONNECTIVES

Relational operators and logical connectives may be used to form complex item-lists and selection-criteria.

Relational Operators may be used in an item-list to constrain the items eligible for processing (refer to the topic titled ITEM LIST FORMATION), or may be used in selection-criteria to limit items to those whose attribute values meet the specified conditions (refer to the topic titled SELECTION-CRITERIA FORMATION). Relational operators apply to the item-id or value immediately following the operator. The absence of a relational operator implies an equality operator.

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

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

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

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

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

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

The usage of the up-arrow and the brackets is further discussed in the topic SELECTION CRITERIA: STRING SEARCHING.

NOTE: These are partial examples and therefore do not illustrate complete ACCESS sentences. They are presented at this point to give the user a general feel for these operators. Complete ACCESS sentences using the above constructs are presented throughout the remainder of the manual.

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

RELATIONAL OPERATORS

CONNECTIVE	MEANING
AND	Both connected parts must be true.
OR	Either connected part must be true. If a logical connective is not given, OR is assumed.

LOGICAL CONNECTIVES

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

Sample Usage of Relational and Logical Connectives

6.9 ITEM-LIST FORMATION

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

6.9.1 EXPLICIT ITEM-LISTS

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

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

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

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

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

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

This means that the ACCESS sentence:

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

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

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

will cause the entire BP file to be searched, with every item-id in the file being matched against the explicit item-list "STAR-TREK".

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

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

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

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

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

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

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

The item-lists may also specify a string-searching capability; this is discussed in the section "SELECTION CRITERIA: STRING SEARCHING".

EXAMPLES:

The SORT verb is used to select and sequence the item-ids in file TEST. (TEST contains 10 items, with item-ids "10" through "19".) The word ONLY used in these examples specifies that only the item-ids are to be listed.

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

PAGE 1                                          15:32:19 20 AUG 1984

TEST........

14
15
16

3 ITEMS LISTED.

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

PAGE 1                                          15:33:01 20 AUG 1984

TEST.......

13
14
15
16
18

5 ITEMS LISTED.
```

SAMPLE USAGE OF EXPLICIT ITEM-LIST

6.9.2 IMPLICIT ITEM-LISTS

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

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

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

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

Other SELECT or SSELECT functions can be used on the implicit list obtained from one SELECT, SSELECT, QSELECT or GET-LIST statement.

EXAMPLES:

```
>SSELECT TEST  [CR]

10 ITEMS SELECTED.

>SAVE-LIST T  [CR]

[214] 'T' CATALOGED, 1 FRAME(S) USED.

>GET-LIST T

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

PAGE 1                                          15:32:19 20 AUG 1984

10
11
12
13

4 ITEMS LISTED.
```

SAMPLE USAGE OF IMPLICIT ITEM-LIST

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

FORMAT:

WITH or IF {NO} {EVERY or EACH} attribute-name {op} {value-list}

Each selection-criterion must begin with the word WITH or IF followed by a single attribute name. (WITH and IF are synonymous.) The attribute name may then be followed by a value-list. The "op" refers to any legal relational operator. The rules for forming value-lists are identical to those for forming item-lists (refer to the topic titled FORMING ITEM-LISTS); double quotes must surround the actual values. For example, the following selection-criterion is met by those items which have at least one value for the attribute DESC which is either equal to "ABC" or is both greater than "DEF" and less than "GHI".

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

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

Several selection-criterion may be bound together by logical connectives to form a complex selection-criterion. When used in this fashion, the AND connective has a higher precedence than the OR connective. A selection-criterion may consist of up to nine "AND clauses". An AND clause is made up of any number of selection-criteria bound by AND connectives. The AND clause is terminated when an OR connective is found in the left to right scan. (NOTE: the absence of an AND connective implies an OR connective.) for an item to pass the selection-criteria, the conditions specified by any one of the AND clauses must be met. An example of the logical hierarchy of AND clauses is shown in the complex selection-criterion below which contains two AND clauses.

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

EXAMPLES:

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

PAGE 1                                              17:36:04  20 AUG 1984

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

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

3 ITEMS LISTED.

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

7 ITEMS COUNTED.

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

PAGE 1                                              17:40:57  20 AUG 1984

ACCOUNT...   TRNS-DATE..

35090        17 MAR 1970
             28 FEB 1970
             17 FEB 1970
             30 JAN 1970
             16 JAN 1970
             29 DEC 1969

END OF LIST
```

SAMPLE USAGE OF SELECTION-CRITERIA

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

String Searching

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

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

The up-arrow (^) indicates a match on any character.

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

EXAMPLES:

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

PAGE 1                                          18:16:56  20 AUG 1984

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

11095       J B STEINER
35065       L J RUFFINE

2 ITEMS LISTED.

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

PAGE 1                                          18:16:56  20 AUG 1984

BP........

$STAR-TREK
*STAR-TREK

2 ITEMS LISTED.
```

SAMPLE USAGE OF STRING SEARCHING SELECTION-CRITERIA

This section is intended to help clarify the actions of the selection processor.

6.12.1 ITEM-ID SELECTION DEFAULT

The ACCESS item-id selection default is the whole file, or the item-ids specified in a list if a list is active.

6.12.2 SELECTION DELIMITERS

In the following examples the delimiters ', ", and \ are all used equivalently to delimit item-ids or values, as the case may be. In general, only the delimiter ' is reserved for item-ids or item-id-related values. The " and \ may be used for values related to data definition item selection criteria and print-limiters as well. They prefer to be associated with the data selection criteria rather than with item-id selection criteria. In general, values delimited by either " or \ will be treated as item-id selection criteria only if they follow the file reference and precede a data definition item.

The delimiter ' will cause the value which it surrounds to be treated as an item-id selection criterion wherever it may be in the sentence.

6.12.3 EXPLICIT ITEM-IDS

If explicit item-ids are specified, then only those item-ids will be returned. If there is a list in effect, it will be ignored.

EXAMPLES:

```
LIST FILENAME 'ITEM1''ITEM2''ITEM3'          or

LIST FILENAME "ITEM1""ITEM2""ITEM3"          or even

LIST FILENAME \ITEM1\\ITEM2\\ITEM3\
```

Each of these will yield a listing of the three items, ITEM1, ITEM2, and ITEM3. The processor does this by retrieving each of these items directly from the file referenced. The collection of explicit item-ids becomes a list, which the processor uses to obtain the next item-id until the list is exhausted, at which time the process terminates.

6.12.4 ITEM-ID TESTS

ACCESS also allows tests on item-ids. All tests are on the item-id as it stands in the file. No conversions or correlatives will be applied to the item-id before the test is made.

The specification of an item-id test rather than the retrieval of a specified set of items is done by including a relational operator, hereinafter referred to as a relational connective, in the item-id specification string. For example:

```
LIST FILENAME 'ITEM1' = 'ITEM2''ITEM3'          or

LIST FILENAME "ITEM1""ITEM2" = "ITEM3"          or even

LIST FILENAME = \ITEM1\\ITEM2\\ITEM3\
```

will all have the same effect. ACCESS will search the whole file, or all the items in a list if there is one in effect, looking for items which have the item-ids ITEM1, ITEM2, and ITEM3. This will take longer than using the explicit item-id reference given above, and is not recommended when you know which item-ids you want.

The intent of relational connectives is to allow specifications of the form

```
LIST FILENAME < 'CAT'
```

which will have the effect of selecting all items which are lexicographically less than CAT, presuming that the file is left-justified. The full effects of justification will be considered below.

Note that in the examples above only one relational connective was included. In that case, all the elements not preceded by a connective are automatically assigned the connective '='. This is true throughout ACCESS in those cases when values are usable.

There need not be spaces between the item or value strings, and the file name, data definition items or connectives may be concatenated to a value in the input sentence, as in the form

```
="ITEM1"
```

In all other cases, all elements in the sentence to be retrieved from a dictionary or dictionary-equivalent file must be surrounded by blanks.

It is possible to specify either a list of item-ids for retrieval or to specify a test on item-ids using this mechanism. It is not possible to retrieve certain items directly and to test all the other items for admissability using only item-id tests. In other words, the item-id list is either a list of explicit item-ids or it is a sequence of values against which to test each item-id in the file. The difference is the inclusion of a relational connective in the item-id list.

6.12.5 ITEM-ID SELECTION CRITERIA

Presume that we have a set of values with an associated relational connective, so that ACCESS is scanning the whole file in order to test the item-ids for acceptability. The item-id test is logically ANDed with all other selection criteria. If the item-id fails the item-id test, the item will be discarded. If one wishes to 'or' an item-id test with other selection criteria, then a data definition item must be included which references the item-id.

EXAMPLES:

Consider the following data definition items.

```
    0          1                        DDI item-id
001 A      001 A                        DDI typifier.
002 0      002 1                        AMC specifier.
003        003                          null label
004        004
005        005
006        006
007        007
008        008
009 L      009 L                        Justification.
010 10     010 10                       Length
```

and the following ACCESS sentences:

LIST MD < "CAT" WITH 1 = "D"	Select item-ids alpabetically less than 'CAT' AND which have a 'D' in attribute 1.
LIST MD < "CAT" AND WITH 1 = "D"	ERROR--Will terminate ACCESS compiler with an error, because the AND connective must be followed by another value which may be ANDed with CAT, and because the AND connective may not immediately precede the first WITH connective in the ACCESS sentence.

When the item-id is referred to as attribute "0", the rules change somewhat as "0" is treated like any other attribute.

LIST MD WITH 0 < "CAT" WITH 1 = "D"	Selects item-ids alpabetically less than 'CAT' OR which have a 'D' in attribute 1.
LIST MD WITH 0 < "CAT" AND WITH 1 = "D"	Selects item-ids alphabetically less than 'CAT' AND which have a 'D' in attribute 1, as in the first case.
LIST MD WITH 0 < "CAT" AND 1 = "D"	This is erroneous. It will have the effect of selecting all items whose item-ids satisfy the CAT criterion, as above. The rest of the sentence has to do with print-limiters.

6.12.6 WITH CONNECTIVE : SELECTION BY DATA VALUE

Attributes may be used as selection criteria in the ACCESS command by preceding the attribute name (or number) with the connective WITH.

FORMAT:

WITH {NOT} {EACH} ATTNAME {value string},

This is called a selection criterion. For convenience, the connectives NOT and EACH following the WITH shall be referred to as selection modifiers.

EXAMPLES:

WITH ATTNAME

will select an item if that attribute has any value other than null.

WITH NOT ATTNAME (or WITHOUT ATTNAME)

will select an item only if that attribute contains only nulls.

WITH EACH ATTNAME

will select an item only if each value in the attribute named is non-null.

WITH NOT EACH ATTNAME (or WITHOUT EACH ATTNAME)

will select any item with at least one null value in the attribute named. The meaning of the term 'value' in this context is considered below.

6.12.6.1 DATA EVALUATION

The selection processor processes the data according to attribute 8 of the data definition item. That is, it executes any conversions or correlatives which are in attribute 8. The result of this calculation is returned to the selection processor. The conversions or correlatives which may be in attribute 7 of the data definition are not applied to data values. The contents of attribute 7, however, may have a significant effect on the success of the selection process as we shall see below.

6.12.6.2 OBTAINING A VALUE (STRING) TO TEST

The selection processor will ignore leading null sub-values within each value. That is, if an attribute of a data item contains multiple values, which themselves contain sub-values, as below,

^\\3\4]6\\7]1\2\3]]\\^

then the processor will retrieve one data value from each value. In this case the values returned will be: 3; 6; 1; null; null, regardless of whether the string was an actual string or a string computed by a F or A correlative.

In either case, if the search for the string results in a null followed by a sub-value mark, the processor will proceed to the next sub value in the string. It will end the data search if a non-null string has been retrieved, or if it encounters a null value followed by a value mark, or if it encounters a null value followed by an attribute mark.

If the search results in a null data value, the process returns to the value test routine. If a non-null data value is returned, the process will then execute any conversions remaining in attribute 8 of the data definition item. The data resulting from the conversion, if any, will then be returned to the value test.

This is the 'value' referred to above. Note particularly that only the first non-null sub-value is returned. If another value is requested, then the next value is taken from the next value, that is, from the right side of the next value mark if there is one. All sub-values which follow the first non-null sub-value are never inspected by the selection processor.

6.12.6.3 EXISTENCE TEST

What occurs at the value test level when testing for existence depends upon the selection modifiers. If there are no modifiers, then if any non-null sub-value is returned from the item, the selection phrase will succeed. If a null value is returned, then the tester will request the next value in the item, unless the last null value was terminated by an attribute mark. In this case, the values within this attribute of this item have been exhausted, and the item does not have the required value. Therefore, this selection phrase fails.

If the selection modifier is NOT, then all values defined by the attribute definition must be inspected in order that the selection phrase succeed. If any value is returned which is non-null, then the clause will fail.

If the selection modifier is EACH, then all values defined by the data definition must be inspected in order to succeed. If any value is returned which is null, then the clause will fail.

If both modifiers are used, WITH NOT EACH or WITHOUT EACH, then the clause succeeds if any value is null. It fails only if all values are non-null.

WITH ATTNAME	succeeds if
(VALUE1 # NULL) OR (VALUE2 # NULL) OR (VALUE3 # NULL) OR ...	
WITHOUT ATTNAME	succeeds if
(VALUE1 = NULL) AND (VALUE2 = NULL) AND (VALUE3 = NULL) AND ...	
WITH EACH ATTNAME	succeeds if
(VALUE1 # NULL) AND (VALUE2 # NULL) AND (VALUE3 # NULL) AND ...	
WITHOUT EACH ATTNAME	succeeds if
(VALUE1 = NULL) OR (VALUE2 = NULL) OR (VALUE3 = NULL) OR ...	

Success conditions for WITH and its modifiers under the test for existence.

WITH ATTNAME	fails if
(VALUE1 = NULL) AND (VALUE2 = NULL) AND (VALUE3 = NULL) AND ...	
WITHOUT ATTNAME	fails if
(VALUE1 # NULL) OR (VALUE2 # NULL) OR (VALUE3 # NULL) OR ...	
WITH EACH ATTNAME	fails if
(VALUE1 = NULL) OR (VALUE2 = NULL) OR (VALUE3 = NULL) OR ...	
WITHOUT EACH ATTNAME	fails if
(VALUE1 # NULL) AND (VALUE2 # NULL) AND (VALUE3 # NULL) AND ...	

Failure conditions for With and its modifiers
under the test for existence.

6.12.7 VALUE STRING

In the syntax of the WITH phrase above, there is an optional value string which has not been mentioned since, although all of the tests for existence assume a null string as the value. A value string is made up of value phrases of the following form

{relational connectives} VALUE.

The relational connectives are optional in the sense that the relation will default to '=' if there is no relational connective preceding the value.

VALUE FORMAT:

"text string"
or
\text string\

Remember that the delimiter ' will always specify an item-id reference.

The contents of the text string may be any characters with the exception of the system delimiters. Avoid the control characters if possible. There are three special symbols, ^, [, and] which have a special meaning to the selection processor, and will be considered below.

6.12.7.1 RELATIONAL CONNECTIVES

The master dictionary contains definitions of the usual relational connectives: =, #, <, >, =>, >=, <=, =<, EQ, NE, GT, GE, LT, and LE. These may be used in any combination except with the condition #, which must be used by itself. Note that all normal combinations are already defined. The form = < may be used as well as =<, for example. Note that the space between the connectives requires that two look-ups must be done, while the =< form is retrieved in a single master dictionary reference. If you have a syntactic preference for the form <>, you may copy the item '#' in the master dictionary to the item '<>'. The operators are logically equivalent.

6.12.8 SPECIFIED VALUES AND ATTRIBUTE 7

It was noted above that the contents of attribute 7 might have an unexpected effect on the results of the selection processor. This is because attribute 7 is generally thought of as an output conversion, because that is what it is designed to affect. For this reason, the ACCESS compiler will execute an inverse conversion on the data values defined in the value string, so that the output conversion does not need to be done for each value in the file referenced by the data definition item. The compiler then throws away the contents of attribute 7.

The ACCESS compiler will not attempt to execute an F- or an A-correlative in attribute 7. These will be ignored.

6.12.8.1 DATE CONVERSIONS

The date conversion will take a date in display format and return the internal form, which is a decimal number representing the number of days since December 31, 1967. In this case you would not wish to execute a date output conversion in attribute 8, since it is unlikely that you would ever get a match. Note that an input date conversion will only transform the external form of a date into the internal form, and that an output conversion will only transform an internal date into the external form of the date. The only time that an input conversion is done in ACCESS is for the evaluation of values associated with selection phrases according to attribute 7.

It is preferable to do the data conversion associated with selection in attribute 7 because it only needs to be done once, at compile time, and because, if it is done in attribute 8 on each value, it will be necessary to remember the precise form which will result, and because the form which derives from attibute 8 will be evaluated according to the alphanumeric form of an external date, rather than in the normally-desired numeric form of the internal date. The internal date is represented as an increasing integer, so that less than and greater than relational connectives have the expected meaning of before and after. The external date does not have this characteristic. It is therefore advisable to store dates in files in the internal form.

6.12.8.2 TIME CONVERSIONS

Time conversions are allowable. Again, it is preferable to use the attribute 7 form. Time is represented in the machine as an integer which is the time since midnight in seconds.

6.12.8.3 MASK CONVERSIONS

In general the forms MR, ML, and MD will treat only the scaling and decimal location characteristics available with these masks. Nothing else will be touched. This means that they will have an effect only if they are immediately followed by one or two numeric digits. They will have an effect only on a value string which represents a number. If the value is a number, it is scaled and the decimal place is attended to. Remember that the internal form is an integer. That is, there is no decimal point in the number. If there are some numeric digits at the front of the value, then these will be taken as the number, and the rest of the value will be thrown away, unless the number is attended to in an attribute-8 F-correlative. If the first character of the specified value is not numeric, then the value string will be taken without modification.

Thus, if attribute 7 has a MR, ML, or MD conversion in it, numeric values are recommended. Note especially that masks of the form 'ML#20' and 'MR#10' have no effect.

6.12.8.4 OTHER MASKING FUNCTIONS

The MCXD (convert from hex to decimal) and MCDX (convert from decimal to hex) conversions have inverse functions, so that they are useable in attribute 7 of a data definition item being used for selection. The MCXD will convert decimal to hex as an input conversion, and the MCXD will convert hex to decimal as an input conversion.

6.12.8.5 TRANSLATE CONVERSIONS

If there is a translate conversion in attribute 7, then it will be executed as an input conversion. This means that the first of the translate attribute mark count numbers in the translate syntax will be used. If the field is null, then the translate will return the item-id if it found an item-id.

If the value specified does not yield an item-id, and if the translate option byte is an 'X', then the value for which the processor will search will be a null. If the option byte is a 'C', then the value for which the processor will search will be the specified value.

What the an input translate will not do is search the file specified by the translate for an item which has the specified value in the correct attribute, and return the item-id as the value.

If there is a direct one-to-one correspondence between the source and destination items, then it is possible to have a set of translate elements within the file which are an inverse transformation. That is, if you supply the value generated by the output translation, and if that value is an item in the file which has as contents the item-id the value which translates to the value supplied, then a translation in attribute 7 is valid. For instance,

In a file we may call CUSTOMER,

```
    232                     Pacific Printing  The item names.
001 Pacific Printing   001 232               The translate references.
```

Then if attribute 7 of the data definition item CUSTTRANS is

```
TCUSTOMER;C;1;1
```

and the data file reference to Pacific Printing is '232', then

```
LIST FILENAME WITH CUSTTRANS = "Pacific Printing" ... CUSTRANS ...
```

will yield the desired result, because Pacific Printing will be translated into 232 for the selection, and 232 will be translated into Pacific printing for output.

A translate which will work in a selection.

If the output translate function translates several different strings to a single output result, such that it would require an inverse function which is multi-valued, then a translate in attribute 7 is inappropriate, because only the first value found by the attribute 7 manipulation will be included in the resultant value string. In this case, the translate must be put in attribute 8, so that the processor is comparing the translated value to the value originally specified in the value string.

6.12.8.6 SELECTION CONVERSIONS : A SUMMARY

It is generally a good idea to use the date and time conversion in attribute 7. The MCXD and MCDX conversions will work. The MR, ML, and MD conversions will work sometimes, and will do strange things other times.

The various other masks and conversions which have no natural inverse functions will tend to fail in a data-sensitive way, and are not recommended.

The processor will not even try to deal with A- and F-correlatives. In all cases the contents of attribute 7 are discarded during the compilation process.

6.12.9 SPECIAL CHARACTERS IN SELECTION VALUES

As noted above, there are three special characters associated with value specification: '[', '^', and ']'. These are not optional and they are not modifiable. They have the following meanings:

[stipulates that any leading string is acceptable.

^ stipulates that any character is acceptable in this position.

] stipulates that any trailing character is acceptable.

The test will terminate at this point with success. For purposes of evaluation, the inclusion of a special character forces evaluation from left-to-right, on a character-by-character basis.

EXAMPLES:

= "[6"	Will accept any data value which terminates in a '6', such as '6' or 'ABC6' or '123456'.
= "3^5"	Will accept any three-character string which begins with a '3' and ends with a '5', such as '305' or '3A5' or '3*5'.
= "6]"	will accept any string which starts with a '6', such as '6' or '6ABC' or '654321'.
= "[6]"	will accept any string which contains a '6', such as '6' or 'ABC6' or '6ABC' or 'ABC6XYZ'.
= "[3^5]"	will accept any string which contains any three-character string which starts with a '3' and ends with a '5', such as '335' or '305XYZ' or 'ABC3X5' or 'ABC3X5XYZ'.

Use of special characters in selection values.

There are certain forms which will not work. If the '[' is used in the value specification, it must be the first character in the string, and it must be the only '[' in the string. If the ']' character is used in the string, it will terminate the specified string at that point. Any characters which may occur after a ']' will never be inspected. The ^ may be used anywhere, and any number of them may be included in the value specification. The form '^^^' may be used to retrieve all three-character strings, for instance, although there is a conversion, 'L', which performs this function.

6.12.9.1 SPECIAL CHARACTERS WITH RELATIONAL CONNECTIVES

The following examples use the relational connectives < (less than), > (greater than) and =, since the other permutations can be derived from these.

The case of equality is shown above. If the form

```
WITH 2 < "5]"
```

is used, the test is on the first character, and is straightforward.

If the form

```
WITH 2 < "[5"
```

is used, the test is on the last character, and is straightforward.

If the form

```
WITH 2 < "[5]"
```

is used, then, if there is a '5' anywhere in the string, equality will be true, and inequality will fail. If there no '5' in the data string, then the condition 'less than' will hold if the last character is less than the 5, and the condition 'greater than' will hold if the last character in the data string is greater than '5'.

If the string of actual data specified is several digits long, the test which generates the type of equality will be as follows: if the process reaches the end of the data string when it is on the first real character of the test string, it will compare those two characters and yield a result as above. If it is on a character other than the first real character in the specified string, it will generate the result expected if the compare were on the first k characters in the specified string against the last k characters in the data string.

Equality will not occur unless all of the real character string is found in the data string.

NOTE:

Truly, many strange and unexpected results can be achieved using certain combinations of special characters and relational connectives, and the user should take great care in the use of such.

6.12.9.2 JUSTIFICATION AND EVALUATION

If numeric data is to sort in proper ascending order, it must be right justified and will print "flush right". Use an "R" on line 9 of the attribute definition item.

Alphabetic data will sort in proper alphabetical order regardless of whether it is right or left justified. Usually alphabetic data is desired "flush left" so an "L" is put on line 9 of the attribute definition.

If the attribute definition item is left-justified or if there is a special character in the value specification string, then comparison will proceed from left to right, and inequality will be declared as soon as characters in the same location in the two strings are different.

The collating sequence is that of the ASCII character set, with the particular characteristic that numbers precede letters, and capital letters precede lower-case letters. An absolute null is less than any character, including an ASCII null. An absolute null occurs when the end of a string is reached, with the result that 'ABC' comes before 'ABC0'.

If the attribute definition item is right-justified, and there are no special characters in the string, and the data string and value string are numeric, then the test will be on the magnitude of the two numbers such that 12 is greater than 2. If these were left-justified, 12 is less than 2 because 1 collates before 2. If the data are not numeric, then they will be compared in the usual left-to-right manner until either inequality is discovered, the strings terminate, or numeric fields are found. If both the data and the specified value are equal up to the start of numeric fields, then the numeric fields will be evaluated as binary integers and compared. If inequality is found, then the string with the smaller imbedded integer is taken as less. If they are equal and both strings terminate at this point, then the strings are equal. If the strings continue with non-numeric data, the left-to-right process continues until inequality occurs or the strings terminate.

In summary, numeric data is normally right justified and alphabetic data is normally left justified.

6.12.9.3 OR CONNECTIVE WITH VALUE PHRASES

It is possible to select based on more than one value. The relation associated with each value is the relational connective which immediately precedes the value. If there is no relational connective which precedes the value, then a default '=' will be inserted into the value string. The implicit relation between the value phrases is 'OR'. If the data value must pass both of two criteria, then there must be an 'AND' between the two value phrases.

EXAMPLES:

The relational connective default:

```
 ... WITH X "A""C""E""G"          is equivalent to

 ... WITH X = "A" = "C" = "E" = "G"

                                  which will succeed if

(DATA = "A") OR (DATA = "C") OR (DATA = "E") ...
```

where DATA in each case represents only one value which may be returned to the value comparison processor.

Therefore we may say,

```
IF ((DATA = "A") OR (DATA = "C") OR (DATA = "E") ...)
```

then DATA IS TRUE else DATA IS FALSE.

A data value is said to succeed if the test returns TRUE.

The cases of inequality are similar:

```
 ... WITH X < "A"  > "C" # "E" <= "G"

                                  is equivalent to

IF ((DATA < "A") OR (DATA > "C") OR (DATA # "E")

     OR (DATA <= "G") ...)
```

Then DATA IS TRUE else DATA IS FALSE.

(This particular case will succeed in all cases).

ORed values with relational connectives.

6.12.9.4 AND CONNECTIVES WITH VALUE PHRASES

To specify a range of values which will be acceptable, or a collection of conditions on a given data value such that they must all be true in order for the condition to succeed, the specified values may be ANDed together.

FORMAT:

... value AND {relational connective} value.

EXAMPLES:

... WITH X <= "A]" AND >= "C]"	will accept all values which start with A, B, or C, as in
IF ((DATA <= "A]") AND (DATA >= "C]"))	
then DATA IS TRUE else DATA IS FALSE.	
... WITH X = "1]" AND < "[5"	will have the effect of accepting all values with start with 1 and end with a character less than 5, as in
IF ((DATA = "1]") AND (DATA < "[5"))	
then DATA IS TRUE else DATA IS FALSE.	

Examples of AND value specification phrases.

6.12.9.5 EVALUATING VALUE PHRASES

An indefinite collection of value phrases may be ANDed together into what we may call an AND value specification phrase. Further, several AND value specification phrases may be ORed together into what we have been calling a value string.

Essentially, an AND phrase, which may consist of sub-conditions, acts as a single entity which can either pass or not pass. For an AND value specification phrase to pass, all elements must pass.

With ORed value specification phrases, if any element succeeds, then the selection criterion succeeds.

6.12.10 SELECTION CRITERIA RELATIONSHIPS

It was just noted that if one ORed value specification allows a data element to pass, the selection criterion will pass. It is possible to have several selection criteria within a single sentence. The default connective between the selection criteria will be an OR, so that if any of the criteria pass, the item will pass. This is of course modified by the selection modifiers NOT and EACH. The NOT modifier will cause the criterion to fail in the case that the value string succeeds and vice versa. We may therefore replicate the table of success and failure under the conditions of WITH, WITHOUT, WITH EACH and WITHOUT EACH which was displayed for the case of existence only. Note that the case of existence is equivalent to the value string # "", although using the explicit string is inefficient. In the table below the form '# NULL' is replaced by the form 'IS TRUE', and the form '= NULL' is replaced by the form 'IS FALSE', as values returned from the value test processor.

EXAMPLES:

WITH ATTNAME <VALUE STRING>	succeeds if
(VALUE1 IS TRUE) OR (VALUE2 IS TRUE) OR (VALUE3 IS TRUE) OR ...	
WITHOUT ATTNAME <VALUE STRING>	succeeds if
(VALUE1 IS FALSE) AND (VALUE2 IS FALSE) AND (VALUE3 IS FALSE) ...	
WITH EACH ATTNAME <VALUE STRING>	succeeds if
(VALUE1 IS TRUE) AND (VALUE2 IS TRUE) AND (VALUE3 IS TRUE) ...	
WITHOUT EACH ATTNAME <VALUE STRING>	succeeds if
(VALUE1 IS FALSE) OR (VALUE2 IS FALSE) OR (VALUE3 IS FALSE) ...	

Success conditions for WITH and its modifiers
under test against a value string.

WITH ATTNAME <VALUE STRING>	fails if
(VALUE1 IS FALSE) AND (VALUE2 IS FALSE) AND (VALUE3 IS FALSE) AND ...	
WITHOUT ATTNAME <VALUE STRING>	fails if
(VALUE1 IS TRUE) OR (VALUE2 IS TRUE) OR (VALUE3 IS TRUE) ...	
WITH EACH ATTNAME <VALUE STRING>	fails if
(VALUE1 IS FALSE) OR (VALUE2 IS FALSE) OR (VALUE3 IS FALSE) ...	
WITHOUT EACH ATTNAME <VALUE STRING>	fails if
(VALUE1 IS TRUE) AND (VALUE2 IS TRUE) AND (VALUE3 IS TRUE) ...	

Failure conditions for WITH and its modifiers
under test against a value string.

6.12.10.1 AND CLAUSES : SELECTION CRITERIA

We may now consider the behavior of the traditional AND clause. Note that there may be a maximum of 9 AND clauses. The sentence will be very difficult to comprehend long before it has acquired 9 AND clauses.

We define the term SELECTION-CRITERION to be of the form

```
WITH {NOT} {EACH} ATTNAME {<VALUE-STRING>}
```

such that each tests one attribute definition item against any value string, and modifies it as specified, across such data values as are available and are required, within one item.

Then an AND clause is of the form

SELECTION-CRITERION AND SELECTION-CRITERION AND SELECTION-CRITERION ....

The criterion for success of an AND clause is that each SELECTION-CRITERION succeed, as per the table above.

6.12.10.2 DATA SELECTION CRITERIA

The data selection criterion is made up of an indefinite number of selection-criteria ORed together. These may include at most 9 AND-clauses and any number of ORed selection criteria which are not members of AND clauses. The condition for success of the data selection criteria is that at least one of the selection criteria which are ORed together succeed.

6.12.10.3 ITEM SELECTION CRITERIA

The condition for item selection is that the item-id tests succeed, and that the data selection criteria succeed. In other words, the item-id test is implicitly ANDed with the data selection test.

6.12.10.4 SELECTION PROBLEMS TO AVOID

The form

```
LIST MD < "CAT" AND WITH 1 = "D"
```

will not work because the item-id test is implicitly ANDed with the data selection criteria, and becuase in this context the AND must either attach an item-id test value to "CAT" or generate an AND clause based on a prior selection criterion. This will generate error message 71.

The form

```
LIST MD < "CAT" OR WITH 1 = "D"
```

will not work because of the implicit ANDing, and has been discussed above.

If you desire the case ((1 = "A" OR 2 = "B") AND 3 = "C"), then it must be written in the following manner:

```
LIST MD WITH 1 = "A" AND WITH 3 = "C" OR WITH 2 = "B" AND WITH 3 = "C"
```

Two data values cannot be compared by the form WITH 1 = 2, because the system has only one temporary data area. If this is desirable, an F-correlative can be generated of the form F;1;2;=, which will return the value 1 when the statement is true, or the value 0 when the statement is false. The above form would be written 'WITH 1=2? = "1"'.

Returning to the relationship between the character surrounding a value and the treatment of the value by the ACCESS compiler, we consider the following example:

```
LIST "20""30""40" FILENAME "50""60" WITH 1.
```

This will compile as though the following had been entered:

```
LIST FILENAME '50''60' WITH 1 = "20" = "30" = "40" = "50 = "60".
```

The values which fall between the file name and the first succeeding attribute definition item will be construed as constituting an item-id test. Since there are no relational connectives associated with these item-id test elements, the process will explicitly retrieve items 50 and 60. It will then test them to see if the data definition item whose name is '1' will return the value 20, 30, 40, 50, or 60. In this case the item will succeed. Otherwise it will fail. This result may be unexpected.

On the other hand, the form

```
LIST '20''30''40' FILENAME '50''60' WITH 1.
```

will behave like the following sentence:

```
LIST FILENAME '20''30''40''50''60 WITH 1
```

Further, the sentence

```
LIST "20""30""40" FILENAME "50""60".
```

will yield the following error message:

```
[19] A VALUE WITHOUT AN ATTRIBUTE NAME IS ILLEGAL.
```

The gist of this is that values delimited by ' will be taken as item-ids or item-id test values, that values delimited by " or \ which fall between the file reference and the first data definition item will be taken as item-ids or item-id test values, and that all other values in the string delimited by " or \ will be associated with either the immediately preceding attribute definition item, if there is one, or with the next attribute definition item, or if there are no attribute definition items in the string, then the sentence will fail in the compiler.

6.13 OUTPUT SPECIFICATION : FORMATION

Output specifications enumerate those attributes to be listed. If no output specifications are specified, the default set of attribute definition items is used to determine the output specifications.

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

If no output specifications are in the ACCESS sentence, the system will use the default output specifications. These default specifications are those contained in the items in the dictionary of the file being listed whose item-ids are the numbers 1, 2, 3, 4, ... The system will sequentially search the dictionary for these items until it comes to an item-id which is not in the dictionary.

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

The LIST, SORT, LIST-LABEL and SORT-LABEL verbs will attempt to format the output into a columnar format with each specified attribute name as a column heading. If line 3 of that specified attribute defining item contains multi-valued "heading text", each multi-value is used on a new heading line, allowing great flexibility in generating multiple line headings.

The number of output columns reserved for each attribute definition item (column width) will be the maximum size from line 10, or the length of the heading in line 3. If the sum of the column widths (adding one blank separator for each specified attribute name) does not exceed the page width as set by the TERM command, then a columnar format will be generated. In a columnar format, the specified attribute names are displayed in a heading across the top of the page. The values for each of the items are then displayed in their respective columns.

If the requested output exceeds the page width, then the attribute names are listed down the side of the output with their respective values immediately to the right. This is known as non-columnar format.

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

Some general forms of OUTPUT-SPECIFICATIONS are exemplified below.

FORMAT:

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

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

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

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

PAGE 1                                              09:09:19  21 AUG 1984

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

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

4 ITEMS LISTED.
```

Columnar Output Format
(Output specifications are underlined.)

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

PAGE 1                                              09:11:53  21 AUG 1984

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

END OF LIST
```

Non-Columnar Output Format
(Output specifications are underlined.)

6.14 PRINT LIMITERS

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

Selection for output of specific values from multi-valued attributes can be accomplished by placing the relational operator(s), and the desired value (or values) in double-quotes (") or backslashes (\), immediately following the attribute name.

If the attribute is an associative controlling attribute then the corresponding values from the dependent attributes will also be returned. Likewise, if the controlling value does not match any of the desired values in quotes, then the dependent values associated with those controlling values will not be printed.

For example, to limit the printing of an attribute called TRAN-DATE (which is a date field) to dates in the range 1/1/84 through 12/1/84 inclusive, the following print-limiting condition may be specified:

```
. . . TRAN-DATE >= \1 JAN 84\ AND <= "12/1/84" . . .
```

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

The string-searching formats may also be used within the print-limiting structure.

Although most practical applications of "print-limiters" will involve multi-valued and/or dependent attributes, this is not to imply that useful "print-limiting" on single valued attributes cannot be done.

The first example lists all the items in the INV file which contain any value for the attribute TRAN-DATE. In the second example, the TRAN-DATE "11 FEB 84" portion of the ACCESS sentence indicates to the ACCESS processor that only the dates equal to 11 Feb 84 are to be retrieved. Since TRAN-DATE is a controlling attribute, only the values associated with the 11th of February for TRAN-TYPE and TRAN-QTY (which are dependent attributes) are retrieved.

EXAMPLES:

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

PAGE 1                                              18:18:36  20 AUG 1984

INV.........   TRAN-DATE   TRAN-TYPE     TRAN-QTY
                          *             *

1242-22        11 FEB      I             100
                           R              48
                           S              31
               12 FEB      I             144
                           R              43
                           S              66
1242-11        11 FEB      I              19
                           R             122
                           S              33
               12 FEB      I              97
                           R              39
                           S               7

2 ITEMS LISTED.
```

Selection of All Values for Multi-Valued Attribute
(Controlling Attribute Name is Underlined.)

```
>LIST INV TRAN-DATE "11 FEB 84" TRAN-TYPE TRAN-QTY  [CR]

PAGE 1                                              18:20:04  20 AUG 1984

INV.....   TRAN-DATE   TRAN-TYPE    TRAN-QTY
                       *            *

1242-22    11 FEB      I            100
                       R             48
                       S             31
1242-11    11 FEB      I             19
                       R            122
                       S             33

2 ITEMS LISTED.
```

Selection of Specific Value for Multi-Valued Attribute.
(Attribute Name and Specific Value are Underlined.)

6.15 DEFAULT OUTPUT-SPECIFICATIONS

If no output-specifications appear in an ACCESS sentence, a set of default attribute definition items are used to determine the output specifications.

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

If an item with item-id "1" cannot be found in the dictionary of the data file specified, a search is made of that users MD for an item with an item-id of "1". If item-id "1" is not found in the MD, then no output specifications are used.

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

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

NOTE:

When listing or sorting DICT items for any file, the attribute defining items will reside in the MD.

When listing or sorting any single level file, again these attribute defining items exist in the MD. A practical example of this would be using ACCESS on the SYSTEM file.

Attribute defining items for the MD reside in that MD.

6.16 SUPPRESSION MODIFIERS

Suppression modifiers are used to cancel some of the default features of ACCESS, namely the heading, columnar headings and item-id listings.

6.16.1 THE ONLY MODIFIER

FORMAT:

ONLY filename

Use of the modifier ONLY just before the file name in an ACCESS sentence will cause the default set of attribute definition items to be ignored. Only the itemnames (item-ids) will display upon output.

6.16.2 THE ID-SUPP MODIFIER (I option)

FORMAT:

ID-SUPP or (I

Item-ids of the data items will appear leftmost on the listing, underneath the file name in the heading, unless the ID-SUPP modifier or 'I' option is used.

6.16.3 THE HDR-SUPP MODIFIER (H option)

FORMAT:

HDR-SUPP or (H option)

The HDR-SUPP modifier (or 'H' option) will suppress the system generated page heading (time and date on the left, page number on the right), and the "n ITEMS LISTED." message at the end of the listing. Note that a HEADING modifier will have this same effect.

6.16.4 THE COL-HDR-SUPP MODIFIER (C option)

FORMAT:

COL-HDR-SUPP or (C

The headers (tags) defined in the dictionary items will appear in the heading, in order from left to right, unless the COL-HDR-SUPP (or 'C' option) is used; a COL-HDR-SUPP will also cause the HDR-SUPP to be in effect.

EXAMPLES:

```
>LIST SM80  [CR]

PAGE  1                                      17:45:10  17 NOV 1984

SM80.......... FRM CLASS.... SUB-CLASS.. REV DATE REV CKSM LINES OBJ

WHERESUBS      121 SYSTEM    UTILITY     25OCT80  80A C5BE   234 1FB
A-CORR1        231 ACCESS    CONVERSION  12SEP80  80A 8BE2    43 0D8
   .

>LIST SM80 HDR-SUPP  [CR]

SM80.......... FRM CLASS.... SUB-CLASS.. REV DATE REV CKSM LINES OBJ

WHERESUBS      121 SYSTEM    UTILITY     25OCT80  80A C5BE   234 1FB
A-CORR1        231 ACCESS    CONVERSION  12SEP80  80A 8BE2    43 0D8
   .

>LIST SM80 (C)  3[CR]



WHERESUBS      121 SYSTEM    UTILITY     25OCT80  80A C5BE   234 1FB
A-CORR1        231 ACCESS    CONVERSION  12SEP80  80A 8BE2    43 0D8
    .

>LIST ONLY SM80  [CR]

PAGE  1                                       23:32:41  14 DEC 1984

SM80..........

WHERESUBS
A-CORR1
BRP1
IDATE
   .
```

Sample usage of Suppression Modifiers.

6.17 MODIFIERS AND OPTIONS

Modifiers or options may be used to further modify the meaning of ACCESS sentences.

A set of modifiers may be used to generate more elaborate listings and reports from ACCESS. Some modifiers appear before attribute names in the ACCESS sentence, and some appear at the end of a sentence, and may be replaced by an option in the option string.

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

MODIFIER	OPTION	MEANING
BREAK-ON		Specifies a break will occur whenever the value of the specified attribute changes.
BY		Designates the attribute name immediately following as a sort-key for the SORT operation. Sequencing is in ascending order comparing ASCII values. values.
BY-DSND		Specifies sorting in descending instead of ascending order.
BY-EXP		Sorts by exploding attribute values; ascending order.
BY-EXP-DSND		Sorts by exploding attribute values; descending order.
COL-HDR-SUPP	(C)	Suppress the output of the time/date heading, the column headings, and the end-of-list message.
DBL-SPC		Causes an extra blank line to be inserted between items to double-space a listing.
DET-SUPP	(D)	Suppresses detail output when used with TOTAL or BREAK-ON modifiers.
DICT		Specifies the DICT (dictionary) portion of a file is to be listed, as opposed to the DATA file.
EVERY or EACH		Modifies a selection-criterion so that every value for a multi-valued attribute must meet the specified condition for the criterion to be true for that item. This modifier must immediately follow the modifier WITH.

FOOTING		Indicates that the following text string is to be processed and used for a footing on the bottom of each page of output.
	(F)	Specifies start of a new page with each item to COPY, LIST-ITEM and SORT-ITEM verbs. (No equivalent modifier.)
GRAND-TOTAL		Indicates the following text is to be printed on grand total lines.
HEADING		Indicates that the following text string is to be processed and used for a heading on the top of each page of output.
HDR-SUPP or SUPP	(H)	Supresses the output of the time/date heading and the end-of-list message.
ID-SUPP	(I)	Suppresses the display of item-ids for LIST and SORT operations.
LPTR	(P)	Routes output to line-printer.
ONLY		Inhibits the use of default attribute definition items when no output specification is indicated. It must precede the file name.
NOPAGE	(N)	When output is to the terminal, this modifier will prevent the halt of output at the end of each page.
TAPE		Causes the data to be obtained from the magnetic tape file in a T-DUMP format.
TOTAL		Causes totals for the attributes which follow to be accumulated.
USING		Causes the attributes to be obtainded from a file other than the dictionary-level file.
WITH or IF		Designates that the following attribute name is part of a selection criterion.
WITHOUT		Is a synonym for WITH NOT or WITH NO.
WITHIN		Designates a special form of processing in which items may contain additional item-ids relating to the primary item-id.

6.18 THROWAWAY MODIFIERS

A, AN, ARE, ANY, FILE, FOR, IN, ITEMS, OF, OR, and THE are throwaway modifiers which may be used to add clarity and naturalness to an ad hoc ACCESS inquiry sentence.

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

EXAMPLES:

```
>LIST THE ACCOUNT FILE   [CR]                                        or
>LIST ACCOUNT  [CR]

>SORT ANY ITEMS IN THE INVOICE FILE WITH A DATE OF "4JUL84" [CR]      or
>SORT INVOICE WITH DATE = "4JUL84"

>SELECT ANY ITEMS IN THE INVENTORY WITH A QTY OF "0"                  or
>SELECT INVENTORY WITH QTY "0" [CR]

>COUNT THE EMPLOYEES WITH A STATUS OF "RETIRED"  [CR]                or
>COUNT EMPLOYEES WITH STATUS "RETIRED"
```

Sample usage of Throwaway Modifiers.

6.19 ACCESS PROCESSOR OPTIONS

Some of the more commonly used modifiers can alternately be invoked by use of the ACCESS OPTIONS feature.

Some ACCESS modifiers may be replaced by OPTIONS. The options string always appears last in the ACCESS sentence and is preceded by a left parenthesis. (The closing parenthesis is optional.) OPTIONS are single alphabetic characters.

Options are decoded during the initial instruction scan. The call to the options processor is initiated by the left parenthesis which must preceed them. The option processor will accept any upper-case alphabetic character. All other characters in the string will be ignored by the options processor for the purposes of ACCESS.

OPTIONS	MEANING
B	Causes the line-feed at the end of the compile phase to be avoided.
C	Equivalent to COL-HDR-SUPP. Relevant to LIST-class verbs.
D	Equivalent to DET-SUPP. Relevant to verbs capable of BREAK-ON and TOTAL.
F	Causes page eject for each item. Relevent only to the LIST-ITEM, SORT-ITEM and COPY verbs.
H	Equivalent to HDR-SUPP. Relevant to LIST-class verbs.
I	Equivalent to ID-SUPP. Suppresses the item-id in LIST-class verbs. Causes the item-id to be output to the terminal with T-LOAD.
N	Equivalent to NOPAGE. Relevant to terminal output with LIST-class verbs.
O	Overwrite items. Relevant to the T-LOAD verb.
P	Equivalent to LPTR. Relevant to LIST-class verbs.

ACCESS OPTIONS.

LIST, SORT, LIST-ITEM and SORT-ITEM ACCESS sentences may use user-designed headings and/or footings.

FORMAT:

HEADING or FOOTING "text {'options'} text {'options'}..."

The HEADING and FOOTING modifiers specify that the following string, (enclosed in double quotes (") or backslashes "\"), is to be used as heading or footing text. Special characters inside the text string, enclosed in single quotes ('), specify special operation on the heading. During generation of the listing, the special characters in the heading or footing text string will be replaced by the current value of the page number, the item-id of the item being listed, break-on data or other parameters. A provision for centering headings, even variable length ones, is provided. The specified heading or footing will be printed at the top or bottom of every page of output.

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

A HEADING (FOOTING) specification may appear anywhere in the LIST type statement.

EXAMPLE:

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

NAME LIST AT 10:29:39  21 AUG 1984
PAGE NO. 1

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

11000               M H KEENER
11015               L K HARMAN
11020               J T O'BRIEN
11025               P R BAGLEY
   .                      .
   .                      .
   .                      .
```

Sample Usage of HEADING

OPTION	MEANING
'B' (Break)	Inserts the value causing a control-break, if the 'B' option has been specified along with the control-break field. This option has no effect otherwise.
'C' (Center)	Causes the current line of the HEADING or FOOTING to be centered on the output page.
'D' (Date)	Inserts the current date at this point in the heading in the form: dd mmm yyyy, where dd is the day of the month, mmm is the abbreviation for the name of the month, and yyyy is the 4-digit year.
'F' (File-name)	Inserts the name of the file being LISTed or SORTed.
'Fn'	(Where 'n' is a decimal number) Causes the file-name to be left-justified in a field of n blanks at that point in the HEADING or FOOTING.
'L' (New line)	Specifies the start of a new line in the HEADING or FOOTING. (Sometimes called the Carriage return/line feed option.)
'P' (Page)	Inserts the current page number, right justified in a field of 4 blanks.
'PN' (Page)	Inserts the current page number,
'T' (Time)	Inserts the current time and date in the form: hh:mm:ss dd mmm yyyy, where hh is the hour in 24-hour (or "military") format, mm is the number of minutes and ss is the number of seconds past the hour, dd is the day of the month, mmm is the abbreviation for the name of the month, and yyyy is the 4-digit year.
''	TWO successive single quotes are used to print a '' single quote mark in heading text.

HEADING and FOOTING Options.

LIST and SORT sentences may generate subtotals and totals for attribute values. The TOTAL modifier is used to generate sub- and grand-totals.

FORMAT:

TOTAL attribute-name {"total limiter"}

The TOTAL modifier causes a total to be computed for the attribute whose name immediately follows the word "TOTAL". On the output, the default total identification is three asterisks (***) in the item-id column.

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

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

The TOTAL modifier may be used in conjunction with the BREAK-ON modifier to output subtotals, as further described in the section SUBTOTALS USING CONTROL BREAKS.

EXAMPLE:

```
>LIST ACCOUNT AFTER "35090" NAME ADDRESS TOTAL DEPOSIT  [CR]

PAGE 1                                          10:31:23  21 AUG 1984

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

35100       R W FORSTROM       318 CARNATION          8.00
35095       A W FEVERSTEIN     324 CARNATION         10.00
35110       H E KAPLOWITZ      306 CARNATION         10.00
35105       S J FRYCKI         312 CARNATION         10.00

                                        ***           38.00

4 ITEMS LISTED.
```

Sample Usage of the TOTAL modifier.

Totals are generated from the number that results after applying the Correlative in attribute 8 of the attribute definition item, and testing for "print-limiters" or "total-limiters".

Totals are generated from the number that results after the execution of attribute 8 of the data definition item, and after the test for the print or total limiter. If the result after the execution of attribute 8 is non-numeric, the total is unchanged.

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

It is possible, however, to use F- or A-correlatives in attribute 7 to generate compositions of totals. In this case the attribute-mark-count number specified in the F-correlative refers to the attribute-mark-count number in attribute 2 of the data defining items. In other words, if an F-correlative stipulates an AMC of 17, at detail-time the processor will look in attribute 7 of a data item for the value requested. At total-time the processor will look for the data defining item in the compiled process controlling string which has '17' as its AMC specifier. This is the number in attribute 2 of the data defining item in the dictionary file. The processor will then retrieve the total related to that item at the current break level for processing in the F-correlative. It is therefore possible to generate data which are the result of arithmentic manipulations on totals.

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

The NB operand returns the break level at which processing is currently occuring. NB is zero at detail-time, and is 255 at grand total time. The lowest level break yeilds 1; and for each succeeding break level the number is incremented by one. The processor is capable of 99 break levels. What this means is that a different class of number may be constructed at eack break level.

The GRAND-TOTAL modifier is used to specify special formatting on the grand-total line.

FORMAT:

GRAND-TOTAL "... text ... {'opTIONS'} ..{text}"

The GRAND-TOTAL connective may be used in reports that have TOTALs and/or BREAK-ONs, to specify special action when printing the grand-total line.

The GRAND-TOTAL connective may appear anywhere within the statement, and it must be followed immediately by a value string enclosed in double-quotes.

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

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

EXAMPLE:

```
>LIST ACCOUNT AFTER "35090" NAME ADDRESS TOTAL DEPOSIT  cs[0][CR]
: GRAND-TOTAL "'U'GRAND-TOTAL IS :" [CR]

PAGE 1                                          10:31:23  21 AUG 1984

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

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

4 ITEMS LISTED.
```

Sample Usage of the GRAND-TOTAL Modifier.

6.23 BREAKING ON ATTRIBUTE VALUES

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

FORMAT:

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

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

During the LIST or SORT operation, a control-break occurs whenever there is a change in the value of the specified attribute. Value comparison is made on a left-to-right, character-by-character basis, with a maximum of the first 24 characters being used in the comparison only. In generating the value for comparison, correlatives in the attribute definition are processed but conversions are not. (See CORRELATIVES and CONVERSIONS).

Up to 15 control-breaks may be specified, the hierarchy of the breaks being specified by the sequence of the BREAK-ONs in the input line, the first being the highest level.

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

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

The BREAK-ON modifier may be used in conjunction with the TOTAL modifier (see SUB-TOTALS USING CONTROL-BREAKS.)

The data associated with the break-on attribute may be suppressed in the detail lines by using a MAX length of zero in the dictionary attribute definition (line 10 in the item). If suppression of the data associated with the control break is desired at total time, it must be done with an F;"" in attribute 7 of th data defintion associated with the control break. R. Additional output formatting capabilities are described in the topic titled OPTIONS FOR CONTROL-BREAKS.

EXAMPLES:

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

PAGE 1                                              09:34:01  02 APR 1977

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

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

                                     ***

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

                                     ***

35030           F M HUGO             DAHLIA            $     123.48
35035           M J LANZENDORPHER    DAHLIA            $     445.89
35040           C E ESCOBAR          DAHLIA            $  38,822.12-
35050           P J WATT             DAHLIA            $     337.18
35055           J W ROMEY            DAHLIA            $  33,478.95

                                     ***

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

                                     ***


***

20 ITEMS LISTED.
```

Sample Usage of Control-Break.
(The BREAK-ON modifier and associated attribute name are underlined)

The TOTAL modifier may be used with the BREAK-ON modifier for the purpose of generating subtotals in LIST and SORT statements when control-breaks occur.

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

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

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

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

In computing the value for accumulation, correlatives are processed but conversion specifications are not (see the section CORRELATIVES AND CONVERSIONS). Conversion is applied only when the value being accumulated is actually printed.

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

PAGE 1                                              09:28:03  22 AUG 1984

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

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

***                                         $ 34,022.34

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

                                ***         $     25.20

***                                         $ 34,047.54

4 ITEMS LISTED.
```

Sample Usage of Control-Breaks
(The BREAK-ON and TOTAL modifiers and
their associated attribute names are underlined)

Headings and output control options may be specified for control-breaks.

FORMAT:

BREAK-ON attribute-name {"text....{' OPTIONS '}....text"}

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

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

The data associated with the BREAK attribute may be suppressed on detail lines if the attribute definition item used with the BREAK-ON modifier has a max-length of zero (line 10).

If OPTIONS are used without accompanying text, they must be enclosed in single quotes within double quotes (e.g. "'V'").

EXAMPLE:

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

PAGE 1                                              11:22:33  21 FEB 1984

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

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

                   SUB-TOTAL FOR 2     $ 34,022.34

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

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

4 ITEMS LISTED.
```

Sample Usage of Control-Break Options
(BREAK-ON modifiers and associated headers are underlined)

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

BREAK-ON OPTIONS.

6.25.1 DET-SUPP MODIFIER

| The DET-SUPP modifier may be used to suppress detail in listings. |

FORMAT:

.....[BREAK-ON and/or TOTAL] ..... DET-SUPP

The DET-SUPP modifier is used with the TOTAL and/or BREAK-ON modifiers. When the DET-SUPP modifier is used with the TOTAL modifier and/or the BREAK-ON modifier, all detail will be suppressed and only the subtotal and total lines will be displayed upon output. Notice that the example ACCESS sentence is the same as the previous section's example, the only difference being the DET-SUPP modifier.

EXAMPLE:

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

PAGE 1                                              09:39:20  22 AUG 1984

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

    SUB-TOTAL FOR         2     $ 34,022.34

    SUB-TOTAL FOR        40     $     25.20

***                             $ 34,047.54

3 ITEMS LISTED.
```

Sample Usage of the DET-SUPP modifier.
(The DET-SUPP modifier is underlined.)

6.26 LIST VERB

LIST is an ACCESS verb which is used to generate a formatted output of selected items and attributes from a specified file.

FORMAT:

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

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

If an item-list is used, items will be listed in the same order as the item-ids appear in the item-list. If no item-list is specified, all items in the file will be listed, and they will appear in order of the group they hash into, and within groups by order of when they were added to the file.

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

EXAMPLE:

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

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

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

EXAMPLE:

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

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

Note that all output from the LIST verb will be to the terminal, unless the LPTR modifier or the " P " option is specified in the ACCESS sentence.

EXAMPLES:

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

PAGE 1                                          11:08:37  12 SEP 1984

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

11115      D R MASTERS          100 AVOCADO           30
11085      A B SEGUR            101 BAY STREET        30
11040      3 G MCCARTHY         113 BEGONIA           30
11050      J R MARSHECK         125 BEGONIA           30
11020      J T O'BRIEN          124 ANCHOR PL         30
11095      J B STEINER          124 AVOCADO           30
11110      D L WEISBROD         106 AVOCADO           30
11015      L K HARMAN           118 ANCHOR PL         30
11105      C C GREEN            112 AVOCADO           30
11090      J W JENKINS          130 AVOCADO           30
23030      L J DEVOS            201 CARNATION         30

11 ITEMS LISTED.

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

PAGE 1                                          11:19:58  13 JUL 1977

ACCOUNT  : 23095
NAME       W E ZUMSTEIN
ADDRESS    224 BEGONIA
START-DATE 01 JAN 1968
DEPOSIT    11.00

ACCOUNT  : 23979
NAME       J W YOUNG
ADDRESS    207 COVE STREET
START-DATE 27 MAR 1970
CURR-BALNC $89.32
DEPOSIT    10.00

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

3 ITEMS LISTED.
```

Sample Usage of the LIST Verb.

6.27 SORT VERB

SORT is an ACCESS verb which is used to generate a sorted and formatted output of selected items and attributes from a specified file.

FORMAT:

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

The output produced by a SORT operation is identical to the output produced by a LIST operation (refer to the LIST VERB), except that a sort operation orders the output in a user-specified order. Sort keys are specified by the BY, BY-DSND, BY-EXP and BY-EXP-DSND modifiers.

6.27.1 BY and BY-DSND MODIFIERS

BY and BY-DSND MODIFIERS

FORMAT:

BY attribute-name or BY-DSND attribute-name

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

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

Sequencing of a SORT operation is accomplished by comparing the ASCII character representations of the attributes specified by the sort keys from left to right, if the attribute is left-justified.

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

6.27.2 CORRELATIVES and CONVERSIONS WITH SORT KEYS

In generating the values used in the sort key comparison, correlatives in the attribute definition are processed, but conversion specifications are not (see section CORRELATIVES AND CONVERSIONS). Also note that several correlative or conversion codes (MR, ML, MC, and D) do not affect the results of sorting and should not be used as correlatives in attribute names which make up sort keys in order to save processing time.

EXAMPLES:

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

PAGE 1                                          14:11:02  22 NOV 1984

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

23000      H T LEE            01 JAN 1968
23005      W B THOMPSON       29 DEC 1969
23010      W E MCCOY          01 JAN 1968
23015      R M COOPER         01 JAN 1968
23020      S L UNGERLEIDER    23 APR 1972

5 ITEMS LISTED.

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

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

11055      W H KOONS          $958,343,75
35080      G A BUCKLES        $447,765.48
11020      J T O'BRIEN        $306,755.54
23040      P B SCIPMA         $123,423.22
23045      P F KUGEL          $ 99,422.34

5 ITEMS LISTED.

>SORT ACCOUNT > "35070" NAME DEPOSIT BILL-RATE [cs]O  [CR]
:BY DEPOSIT BY BILL-RATE  [CR]

PAGE 1                                          14:15:47  25 OCT 1984

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

35090      D U WILDE             3.17     10.03
35100      R W FORSTROM          8.00     10.03
35080      G A BUCKLES          10.00       .35
35095      A W FEVERSTEIN       10.00       .35
35105      S J FRYCKI           10.00       .35
35075      J L CUNNINGHAM       10.00       .40
35085      J F SITAR            12.00       .02

7 ITEMS LISTED.
```

Sample Usage of the SORT Verb with Sort Keys.

6.27.3 BY-EXP and BY-EXP-DSND MODIFIERS - EXPLODING SORTS

The EXPLODING SORT on multi-values allows the system processors to access individual values in a multi-valued item, and to sort that item according to any user specified value (or values).

FORMAT:

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

The explosion-limiter is of the same form as the print-limiter (see section PRINT LIMITERS), and serves to limit the explosion to only those multi-values that pass the limiting condition.

The exploding sort capability is used to "explode" an item into effectively multiple items, the explosion being controlled by a multi-valued attribute or attributes in the data.

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

The explosion is caused by using the BY-EXP modifier instead of the BY modifier (or the BY-EXP-DSND instead of the BY-DSND). The attribute that follows the BY-EXP may be multi-valued; there will be as many pseudo-items created as there are values in the attribute. If multiple BY-EXPs are specified, the attribute with the maximum number of multiple values will be used to create the items, with other fields being treated as null if there is no data for all the values.

A single-valued attribute that is specified in the output specification of the SORT using an exploding modifier will have the single value repeated in each occurrence of the sort key.

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

A Publisher's mailing-list file is comprised of items which contain the names, addresses, special type code and subscription dates of all customers. The Item-Id is a last name concatenated with a first initial and the item contains the above information for all customers with that last name and first initial. Attributes in the item are ordered as follows:

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

Each attribute contains a number of values; one value per customer. For the sake of demonstration, we will assume the file 'MASTER' contains only the following 2 items:

```
Item-Id :  SMITH*J
001 JOHN T]JIM W]JANET]JOSEPH K
002 11 NORTH]BOX 301] 77 SUNSET]405 NASTER
003 L.I. NY]PLAINS GA]MIAMI FL]IRVINE CA
004 37901]44506]22116]22116]33288
004 4D]7D]9E]3E
005 6/66]8/72]4/76]11/75

Item-ID :  JONES*T
001 TOM F]TERRY]TEDDY]TIM
002 1 APPLE]56 FIRST]45 HOLLY]112 ELM
003 AKRON OH]MODESTO CA]TAMPA FL]JACKSON MS
004 44300]33299]2117]98761
005 6D]2D]7D]5D
006 4/75]3/76]11/71]4/73
```

Thus a master address listing sorted by zip code can be obtained with the following Access statement:

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

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

FIRST NAME	LAST NAME	ADDRESS	STATE	ZIP	CODE DATE
JANET	SMITH	77 SUNSET	MIAMI FL	22116	9E 4/76
TEDDY R	JONES	45 HOLLY	TAMPA FL	22117	7D11/71
JOSEPH K	SMITH	405 NASTER	IRVINE CA	33288	3E11/75
TERRY	JONES	56 FIRST	MODESTO CA	33299	2D 3/76
JOHN T	SMITH	11 NORTH	L.I. NY	37901	4D 6/66
TOM F	JONES	1 APPLE	AKRON OH	44300	6D 4/75
JIM W	SMITH	BOX 310	PLAINS GA	44506	7D 8/72
TIM	JONES	112 ELM	JACKSON MS	98761	5D 4/73

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

The exploding-sort modifiers may also be used with explosion limiters. For example, the Access statement: >SORT MASTER BY-EXP ZIP > "39999" ID-SUPP FNAME LNAME ADR1 ADR2 ZIP CODE DATE

yields only values whose corresponding zip is greater than 39999:

FIRST NAME	LAST NAME	ADDRESS	STATE	ZIP	CODE DATE
TOM F	JONES	1 APPLE	AKRON OH	44300	6D 4/75
JIM W	SMITH	BOX 310	PLAINS GA	44506	7D 8/75
TIM	JONES	112 ELM	JACKSON MS	98761	5D 4/73

Sample Usage of EXPLODING SORTS

6.28 WITHIN CONNECTIVE

The WITHIN connective used with a LIST or COUNT verb can retrieve and list all of the items which are sub-items of a specified item.

FORMAT:

```
LIST WITHIN file-name 'item-id' {options ... }
```

The WITHIN connective may be used to list or count the tree explosion of one item that contains an attribute which contains one or more (multivalued) values that are, in turn, item-ids within the same file. This second level item may also contain, in the same attribute number, one or more item-ids. The explosion may proceed up to 20 sub-levels. The DL/ID of the file must have a V code in attribute 8.

ATTRIBUTE 8 FORMAT:

```
V;;attribute-no.-to-explode
```

As an example, this capability is useful for bill-of-material processing, where you have assemblies and sub-assemblies.

```
>LIST DICT ASSEMBLY 'ASSEMBLY' [CR]

PAGE 1                                          08:39:17  01 JUL 1984

ASSEMBLY  D/CD    AMC      S/NAME   STRUCT     CORR      TYP       MAX

ASSEMBLY  D       34012    23       1          V;;2      L         10

1 ITEMS LISTED.
```

Sample Dictionary V code entry

```
>LIST WITHIN ASSEMBLY 'A200-123' PART# DESC SUB.ASS QOH (I) [CR]

PAGE 1                                      11:08:37  12 SEP 1984

LEVEL    PART NO.   DESCRIPTION              SUB.ASS    ON-HAND

1        A200-123   SERVOS                   A201-789      53
                                             A201-890
2        A201-789   D.C.MOTOR                A202-101      24
                                             A202-102
3        A202-101   D.C.MOTOR PLATFORM                     73
3        A202-102   D.C.MOTOR POWER UNIT                   54
2        A201-890   SERVO BOARD                            12

5 ITEMS LISTED.
```

Sample Usage of WITHIN connective.

LIST-LABEL and SORT-LABEL are ACCESS verbs that may be used to print mailing labels or to produce other special purpose listings.

FORMAT:

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

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

(See the LIST and SORT verbs for parameter information.)

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

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

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

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

These parameters determine only the arrangement of attribute data into lines of labels, and are entered in the following format:

PARAMETER	MEANING
count	The number of items (labels) across each page
rows	The number of lines printed for each label (height of each label, in rows)
skip	The number of lines to skip between labels (vertical spacing between labels, in rows)
indent	The number of spaces to indent the data from the left margin
size	The maximum width allowable for the data associated with each attribute name (width of each label, in columns)

space	The number of horizontal spaces between items (horizontal spacing between labels, in columns)
C	Optional; if present, specifies that null attributes are not to be printed. If the C is not specified, null values will be printed as all blanks.

Values must conform to the range:

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

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

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

where "parameter" is the invalid numeric parameter entered.

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

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

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

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

PAGE 1                                        11:22:33  25 OCT 1984

CUST.NAME.    AARON AARONSON      ABE AARONSON        CITY OF CARSON
ADDRESS...    1213 N. ELAINE      18286 FOXGLOVE      P.O. BOX 9905
CITY/STATE    MELBROOK, OHIO      BOSTON, MASS.       CARSON, CALIF.
ZIP CODE..             34523              10042               92412

CUST.NAME.    JACK JOHNSON        KELLY JOHNSON       LARRY JOHNSON
ADDRESS...    845 AVOCADO         20650 MARCHETA      525 DUNNEGAN
CITY/STATE    FLINT, ARIZ.        SURF, NEW MEXI      BOSTON, MASS.
ZIP CODE..             43321              54000               10042

6 ITEMS LISTED.
```

Sample Usage of SORT-LABEL Verb.

6.30 THE REFORMAT AND SREFORMAT VERBS

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

FORMAT:

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

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

NOTE: exploding sort keys are valid, but control breaks and totals are not. The REFORMAT and SREFORMAT verbs function almost identically with the LIST and SORT verbs, the only difference being that the output is not made into a listing. Instead, the output can be used as data to update items in a file, or to write tape records.

If the PROC stack is not on, the user's terminal will be prompted with:

```
FILE NAME:
```

at this point, the user enters the name of the destination file, where the output of the list processor will be stored, or enters the word TAPE for magnetic tape. The destination file name is entered either through the user's terminal, or through the Secondary Output Buffer (Stack) in PROC. A null response will indicate that the file specified in the ACCESS sentence will be used as a destination file.

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

REFORMATTING TO ANOTHER FILE

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

REFORMATTING TO MAGNETIC TAPE

When reformatting a file to mag tape, the values are concatenated together, and either truncated or padded at the end with nulls (hex '00's) to obtain a record the same length as the current tape record length, as specified by the T-ATT verb.

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

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

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

>REFORMAT EMPLOYEE NAME SSN [CR]

FILE NAME:NAMES [CR]

>SORT NAMES SSN HDR-SUPP [CR]

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

Example Use of REFORMAT Verb to Create a Cross-Index.

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

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

     12345 JACKSON ENTERPRISES  523-3888 P.O. BOX 322 JACKSON, MISS.
     12888 ACME BOLT CO.        444-8819 17911 SKY PARK CIR IRVINE, CA.

>T-ATT (80) [CR]          (Specifies 80-byte tape records.)
TAPE ATTACHED
>SREFORMAT VENDORS NAME PHONE ADDRESS [CR]

FILE NAME:TAPE [CR]

    1 12345
    2 12888
2 ITEMS DUMPED.

>T-REW [CR]
>T-READ [CR]

L 0050 12:18:15  28 JUN 1984 VENDORS

RECORD = 1

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

RECORD = 2

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

[94]  END OF FILE
```

Output to Magnetic Tape from ACCESS Using SREFORMAT Verb.

COUNT is an ACCESS verb which counts the number of items meeting the conditions as specified by the combination of item-list and selection-criteria.

FORMAT:

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

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

The COUNT verb will generate the error message :

```
n ITEMS COUNTED.
```

where "n" is the number of items meeting the specifications set down by the item list and selection criteria, if present. If neither item list nor selection criteria are specified, the number (n) returned will be the number of items in the specified file.

```
>COUNT MD WITH 1 = "P]"                    Count verbs and procs in your MD.
126 ITEMS COUNTED.

>COUNT TEST  [CR]                          Count all items in file TEST.
10 ITEMS COUNTED.

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

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

Sample Usage of COUNT Verb.

SUM is an ACCESS verb which generates the total of all the values of one attribute name for a selected set of items in a file.

FORMAT:

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

SUM VERB OUTPUT DISPLAY:

```
TOTAL OF aaaa IS : xxxx
```

where aaaa is the header for the attribute name, (the header from line 3, or the item-id if line 3 is null), and xxxx is the computed total for the attribute name.

NOTE: correlatives are processed in determining totals for the SUM verb, but conversions are not processed. Any conversions present will be applied just before the total is printed. (See CORRELATIVES AND CONVERSIONS).

```
>SUM ACCOUNT CURR-BALNC  [CR]
TOTAL OF CURR-BALNC IS :  $2,405,118.10
>SUM ACCOUNT CURR-BALNC WITH CURR-BALNC > "100000"  [CR]
TOTAL OF CURR-BALNC IS :  $1,836,287.99
>SUM ACCOUNT > "35055" CURR-BALNC  [CR]
TOTAL OF CURR-BALNC IS :  $605,916.48
>SUM DICT ACCOUNT V/MAX  [CR]
TOTAL OF V/MAX IS :  2887
>SUM ACCOUNT DEPOSIT WITH CURR-BALNC <"50000"& WITH NO SEWER-ASMT  [CR]
TOTAL OF DEPOSIT IS :  460.00
```

Sample Usage of SUM Verb

STAT works identically to SUM, but STAT also provides a count of the number of items selected and an average value for the attribute name.

FORMAT:

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

STAT VERB OUTPUT DISPLAY:

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

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

NOTE: correlatives are processed in determining totals for the STAT verb, but conversions are not processed. Any conversions present will be applied just before the total is printed. (See CORRELATIVES AND CONVERSIONS).

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

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

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

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

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

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

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

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

Sample Usage of STAT Verb

6.34 THE SELECT AND SSELECT VERBS

SELECT is an ACCESS verb which provides the facility to select a set of items or attribute values from a file, and generating a select-list of the selected item-ids or values. The SSELECT verb combines the SORT capability with the SELECT capability.

FORMAT:

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

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

The difference between the SELECT and LIST verbs (or between the SSELECT and SORT verbs) is in the handling of the attribute name output as specified by the output specifications. With the LIST or SORT verbs, the output for each attribute name is directed to the user's terminal, a line printer, a magnetic tape unit, or a SPOOLER hold file. With the SELECT or SSELECT verbs, the attribute values and/or item-ids in the output specifications are saved in a select-list. This "select-list" is a temporary list which is stored in the user's workspace until the execution of one more verb. The next verb executed will use the select-list as its implicit item list. The output from a SELECT or SSELECT sentence is the select-list and error message 404:

```
n  ITEMS SELECTED.
```

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

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

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

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

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

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

A select-list may be permanently saved by use of the SAVE-LIST verb, or it may be passed as a parameter to PICK/BASIC or RUNOFF.

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

PICK/BASIC The select-list is available to the PICK/BASIC program via the READNEXT statement. The select-list will override the first SELECT statement in a PICK/BASIC program. (See PICK/BASIC.)

ACCESS The select-list is available to ACCESS verb processors as an implicit item-list, and will override any item-lists in the ACCESS sentence. The selection criteria are still processed. (See FORMING ITEM-LISTS.)

RUNOFF The selected item-ids or values may be inserted into RUNOFF text by use of the READNEXT command.

SAVE-LIST The select-list may be cataloged into the POINTER-FILE by use of the SAVE-LIST verb. This saved select-list is now available to any process on any line through the GET-LIST verb. (See GET-LIST and SAVE-LIST.)

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

Some of the available disc space will be used to store the temporary select-list. This space will be released after the select-list has been processed.

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

11 ITEMS SELECTED.

>EDIT ACCOUNT  [CR]
     .
     .

>SSELECT ACCOUNT > "11045" WITH CURR-BALNC LE "0"  [CR]

3 ITEMS SELECTED.

>RUN BP ACCOUNT.ZERO
```

Sample Usage of Select and SSelect Verbs.

6.35 THE SAVE-LIST, GET-LIST, AND DELETE-LIST VERBS

The verbs SAVE-LIST, GET-LIST and DELETE-LIST are used to save, retrieve, and delete selected item-lists.

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

The SAVE-LIST verb provides the facility to make a permanent select-list out of a temporary select-list produced by the SELECT, SSELECT and QSELECT verbs. These permanent select-lists are retrievable via the GET-LIST verb, and may be deleted via the DELETE-LIST verb. Using the list processor requires an account to have a file called POINTER-FILE with a DC on line 1 of its 'D' pointer in that account's MD.

FORMAT:

```
SAVE-LIST  list-name
```

The SAVE-LIST verb will catalog the select-list, (save it in overflow frames) and add or update the pointer to the select-list in the POINTER-FILE with the item-id of the assigned list-name. The system will respond with message 241:

```
[241] 'list-name' CATALOGED;  n FRAMES(S) USED.
```

where "n" is the number of frames of overflow needed to store the list. A previously existing catalogued select-list with the same name will be overlaid by the newly catalogued select-list.

If a SAVE-LIST command is entered at TCL level, it must immediately follow the SELECT or SSELECT sentence that generated the desired temporary select-list. If the SAVE-LIST sentence is entered in a PROC, it must be placed in the PROC Secondary Output Buffer (Stack). Files other than the POINTER-FILE may be created for saving lists. They must contain a 'DC' on line 1 of their pointer. To use these files requires the format:

```
SAVE-LIST {DICT} file-name list-name
```

The GET-LIST verb looks up a pointer in the POINTER-FILE and retrieves the select-list to which it points.

FORMAT:

```
GET-LIST {file-name} list-name
```

The list-name specifies which saved list is requested. A POINTER-FILE item-id will be generated exactly as for a SAVE-LIST verb, and the POINTER-FILE item will be used to find the saved select-list. The file-name is specified if not using the POINTER-FILE. If the item cannot be found, the system will respond with message 202:

```
[202] 'list-name' NOT ON FILE.
```

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

n ITEMS SELECTED.

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

If the next verb after a GET-LIST is entered in a PROC, it must be entered in the PROC Secondary Output Buffer (Stack).

The DELETE-LIST verb removes a pointer to a select-list from the POINTER-FILE and returns the frames containing the catalogued list to the overflow pool.

FORMAT:

DELETE-LIST {file-name} list-name

A POINTER-FILE item-id will be generated, just as for a SAVE-LIST verb, and the item will be looked up in the POINTER-FILE, unless file-name is specified. If the pointer item cannot be found, the system will respond with error message 202:

[202] 'list-name' NOT ON FILE.

where 'list-name' is the name following the GET-LIST verb, or null, if no name was specified. If the item is found, the system will respond with error message 242:

[242] 'list-name' DECATALOGED.

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

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

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

[241] 'OVER.35' CATALOGED, 1 FRAME(S) USED.

>GET-LIST OVER.35  [CR]

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

24 ITEMS COPIED.

>GET-LIST OVER.35

24 ITEMS SELECTED.
>RUN BP BILLING-OVER.35

>DELETE-LIST OVER.35  [CR]
[242] 'OVER.35' DECATALOGED.
```

Sample Usage of SAVE-LIST, GET-LIST, and DELETE-LIST.

6.36 THE COPY-LIST, EDIT-LIST AND QSELECT VERBS

The verb COPY-LIST enables the user to copy a saved select-list to the terminal, another select-list, or to an item in a file. The EDIT-LIST verb allows the editing of a saved select-list. The QSELECT verb allows the creation of a select-list from attributes in an item or items in a file.

FORMAT:

COPY-LIST {list-name {account-name} } { (options) }

where list-name and account-name are as described in the SAVE-LIST section. The options can be:

OPTION	DESCRIPTION
D	When copying a select-list to another list, the original select-list is Deleted after the copy.
N	When copying a select-list to the terminal, the automatic end-of-page wait is inhibited.
P	The select-list is copied to the line-printer.
T	The select-list is copied to the terminal.
X	On a terminal or line-printer copy, the data is displayed in hexadecimal.

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

TO:

The general response to this message is of the form:

{ (File-name } list-name

If the list-name form is specified, the original select-list is copied to the newly specified list-name. If the list-name is not specified, a null list-name is assumed. The original select-list will be deleted if the D option had been specified. Note that the new select-list will overwrite any existing select-list with the same list-name. Also note the use of a left parenthesis preceding the use of a different file-name to copy to.

If the form with the file-name is used, the select-list will be converted to standard item format, with each element of the select-list being stored as an attribute. If the resulting item exceeds the maximum size of 32,267 bytes, it will be truncated at that point.

The EDIT-LIST statement may be used to edit a previously saved select-list.

FORMAT:

EDIT-LIST {file-name} list-name

The system EDITOR will be entered; each element in the select-list will be treated as a line in the EDITOR. All normal EDITOR functions, including MERGE, are valid. (See EDITOR.)

The QSELECT verb is used to generate a select-list from attribute(s) within an item or items in a file.

FORMAT:

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

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

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

Just as with any other verb of this format, the item-list may be omitted by preceding the QSELECT statement with a SELECT, SSELECT, GET-LIST or another QSELECT statement.

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

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

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

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

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

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

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

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

>QSELECT SYSPROG-PL COLD-LIST [CR]

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

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

Examples of COPY-LIST, EDIT-LIST and QSELECT usage.

ISTAT is an ACCESS verb which provides file utilization information.

FORMAT:

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

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

The average number of bytes per group is an excellent indicator as to how well the file space is being utilized. If each group had less than 500 bytes in it, no additional frames of overflow would be needed for the file. (See first example.) However, if every group had 501 bytes in it, an additional frame would be linked to each group in the file, just to contain that one last byte, and the remaining 499 bytes in the linked frame would be wasted. (See second example.)

```
>ISTAT TEST-FILE  [CR]

FILE= TEST-FILE MODULO= 29                              16:11:30  27 DEC 1984
BYTES ITMS
05644 023 *>>>>>>>>>>>>>>>>>>>>>>
05645 022 *>>>>>>>>>>>>>>>>>>>>>>
05646 018 *>>>>>>>>>>>>>>>>>>
  .
  .
05671 017 *>>>>>>>>>>>>>>>>>
05672 025 *>>>>>>>>>>>>>>>>>>>>>>>>
05673 020 *>>>>>>>>>>>>>>>>>>>

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

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

FILE= TEST-FILE MODULO= 29                              16:11:30  27 DEC 1984
BYTES ITMS
05644 003 *>>>
05645 001 *>
05646 000 *
  .
  .
05671 002 *>>
05672 004 *>>>>

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

Sample Usage of ISTAT Verb.

6.38 HASH-TEST VERB

HASH-TEST is an ACCESS verb which provide file utilization information. The HASH-TEST verb can be used to determine the best modulo for a given file. It shows how any or all of the items in the file would hash into groups, for any given modulo, as well as the other figures and averages provided by the ISTAT verb.

FORMAT:

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

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

```
TEST MODULO:
```

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

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

FILE= JUNK-1  MODULO= 3                                  12:33:13  22 FEB 1984
BYTES ITMS
00400 001 *>>>
00455 001 *>>>
00345 001 *>>>

ITEM COUNT=          9, BYTE COUNT=     1200, AVG. BYTES/ITEM=   133.3
AVG.ITEMS/GROUP=  3.0, STD. DEVIATION=    .0, AVG. BYTES/GROUP=  400.0
     (An example showing desirous file utilization.)

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

FILE= JUNK-2  MODULO= 3                                  12:33:13  22 FEB 1984
BYTES ITMS
00600 001 *>>>
00655 001 *>>>
00845 001 *>>>

ITEM COUNT=          9, BYTE COUNT=     2100, AVG. BYTES/ITEM=   233.3
AVG.ITEMS/GROUP=  1.0, STD. DEVIATION=    .0, AVG. BYTES/GROUP=  700.0
     (Note that now there must be 2 frames in each group!)
```

Sample Usage of HASH-TEST Verb.

T-DUMP is an ACCESS verb which dumps a specified file to magnetic tape. T-LOAD is used to restore data form a previously generated T-DUMP tape. The TAPE modifier may be used to list or otherwise access data from a T-DUMP tape.

T-DUMP

An ACCESS sentence using the T-DUMP verb may specifiy selection criteria, but not output specifications.

FORMAT:

```
T-DUMP {DICT} file-name {item-list} {selection-criteria}
       {HEADING "text"} {modifiers} { (options,options,...options) }
```

The magnetic tape unit must be attached by the user before the T-DUMP command is issued. The T-ATT command will also setup the tape record length to be used in the T-DUMP. (See T-ATT.)

T-DUMP causes a tape label to be written on the magnetic tape drive, followed by a dump of the selected items. If the optional HEADING and text are specified, the text is added to the standard heading, which is of the form "{DICT} file-name". If the optional DICT is included, the dictionary section of the file will be dumped, and no File Definition Items (with line 1 = "D", "DX", "DY" or "DC") will be dumped. An EOF (End-Of-File) mark is written to the tape after the dump.

The HDR-SUPP connective or (H) option is used to suppress the tape label.

The ID-SUPP connective, or the (I) option is used to suppress the listing of item-ids that are being dumped.

T-LOAD

An ACCESS sentence using the T-LOAD verb may specifiy selection criteria, but not output specifications.

FORMAT:

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

The magnetic tape unit must be attached by the user before the T-LOAD command is issued.

T-LOAD causes the tape label to be read form the tape; this will also setup the tape record length from the label. If unlabeled tapes, or non-PICK tapes are used, the appropriate tape record length must be setup in the T-ATT statement.

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

If the ID-SUPP connective, or the (I) option is used, the listing of item-ids being loaded will be suppressed.

The tape will be positioned at the EOF marker at the conclusion of the load; this will happen even if the T-LOAD is aborted by using a BREAK and END sequence. (See also PERIPHERALS, T-DUMP & T-LOAD)

THE TAPE MODIFIER

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

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

31 ITEMS DUMPED

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

>T-DUMP TEST-FILE  [CR]

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

5 ITEMS DUMPED

  This sentence dumps the entire TEST-FILE file to the magnetic tape.
```

Sample Usage of T-DUMP Verb.

EXAMPLES:

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

4 ITEMS LOADED.

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

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

17 ITEMS LOADED

 This sentence loads only those items from the T-DUMP tape that
 have item-ids in the range 100 through 400.
```

Sample Usage of T-LOAD Verb.

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

 This sentence will use the dictionary of the TEST-FILE to gen-
erate the default output specifications; then will read the
T-DUMP tape and format the data items it finds there in the
standard listing format.

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

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

Sample Usage of TAPE modifier.

6.40 THE LIST-ITEM AND SORT-ITEM VERBS

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

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

FORMAT:

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

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

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

OPTION	MEANING
N	If output is to terminal, inhibits the wait at the end of each page (NOPAGE).
P	Route output to lineprinter (LPTR).
F	Causes a Form-feed for each item. Starts a new page for each item.
S	Supresses line numbers.

(Equivalent connectives in parentheses)

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

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

>SORT-ITEM MD WITH 1 "PQ" HEADING "PROC: 'I10' PAGE'P'" (P,F)  [CR]

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

Sample Usage of the LIST-ITEM and SORT-ITEM Verbs.

6.41 CONTROLLING AND DEPENDENT ATTRIBUTES: AN INTRODUCTION

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

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

The controlling-dependent set must be maintained in a particular format by whatever program is updating the file; see the FILE STRUCTURE section for a general discussion on the physical item format and the usage of value delimiters.

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

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

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

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

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

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

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

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

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

 ACC..............  DATE....  TIME....  CONN...  UNITS  LPTR
                              *         *        *      *
 SMITH#0            03/01/84  10:33AM   00:10    222   11          1
                              04:15PM   00:05     23               1
                    03/03/84  11:12AM   00:33   1123   78          2
                              12:13PM   00:04     34               2
                              01:01PM   01:03   3090    6          2
```

There are two dates in the Controlling attribute, 03/01/84 and 03/03/84; associated with these two dates are two sets of values in the attributes TIME, CONN, UNITS and LPTR. (These are marked as "1" and "2" on the far right). The data under the DATE column are stored as two multi-values; that under the other columns are also two multi-values, but each is further comprised of sub-multi-values, two in the first set and three in the second.

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

```
SMITH#0[am][am][am][am]03/01/84[vm]03/03/84[am]
                      ..attribute 4........

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

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

11[Vm]78[svm][svm]6[am]
 .. Attribute 7 ........

222[Svm]23[vm]1123[svm]34[svm]3090[am]
 .. Attribute 8 ......................
```

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

'C' CODE FORMAT:

C;amc;amc;...amc

where 'C' is the capital letter C, and each 'amc' is a dependent attribute number.

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

'D' CODE FORMAT:

D;amc

where 'D' is the capital letter D, and 'amc' is the single attribute number of the attribute which controls the dependent attribute.

EXAMPLE:

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

The dictionary of the CHECK-REGISTER file looks like:

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

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

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

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

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

JAN            501              $123.00 01 JAN 1984
               502                $4.00 03 JAN 1984
               503               $44.88 03 JAN 1984

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

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

JAN
               502                $4.00 03 JAN 1984
```

Sample Use of Controlling and Dependent
Structure-definition Codes (With Print Limiting)

6.43 SUMMARY OF CONVERSION AND CORRELATIVE CODES

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

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

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

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

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

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

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

Conversions and correlatives may be multi-valued, in which case they are separated by a value-mark (VM = control shift M = Hex 'FD').

NAME	DESCRIPTION
A	ARITHMETIC. Used to compute mathematical expressions. Converted to an "F" code at run-time.
C	CONCATENATE. Used to concatenate attribute values.
D	DATE. Used to convert dates to external format.
F	FUNCTION. Used to compute a mathematical function on attribute values.
G	GROUP. Used to extract one or more fields seperated by a given delimiter. delimiter.
L	LENGTH. Used place constraints on what kind of data will be returned, based on the length.
MC	MASK CHARACTER. Used to convert strings to upper or lower case, or to extract alphabetic or numeric characters from strings.
ML	MASK DECIMAL. (Left justified) Used to format and scale numbers and dollar amounts (same as PICK/BASIC format string).
MR	MASK DECIMAL. (Right justified) Same as ML.
MT	MASK TIME. Used to convert time of day from internal to external format.
MX	MASK HEXADECIMAL. Used to convert ASCII character strings to their hexadecimal (Base 16) representations.
P	PATTERN MATCH. Used to return only those data values which match the specified pattern.
R	RANGE. Used to return only those data values which fall within the specified ranges.
S	SUBSTITUTION. Used to substitute the data value for none null or zero values.
T	TEXT EXTRACTION. Used to extract a fixed field from an attribute value.
Tfile	FILE TRANSLATION. Used to convert attribute values by translating them through another file.
U	USER-DEFINED. Used to evoke assembly language routines to perform special user-written conversions or correlatives.

Correlative and Conversion Processing Code Summary.

6.43.1 'G' CODE : CORRELATIVE AND CONVERSION GROUP EXTRACTION CODE

The 'G' code is used to extract from a value, one or more fields, separated by a given delimiter.

FORMAT:

G{m}*n

PARAMETER DEFINITIONS:

G	is the group extraction code.
m	optionally specifies the number of fields to skip. If m is not specified, zero is assumed, and no fields are skipped.
*	Represents any single non-numeric character which is the field separator, except a minus-sign (-), or any system delimiter (SM, AM, VM, SVM or SB).
n	is a decimal number which is the number of contiguous fields to be extracted.

If an attribute value consists of multiple fields separated by a delimiter, the 'G' code can be used to extract one or more contiguous fields.

EXAMPLES:

G Code	Attribute Value	Output Value
G/1	04/02/1984	04
G1/1	04/02/1984	02
G2/1	04/02/1984	1984
G/2	04/02/1984	04/02
G1/2	04/02/1984	02/1984
G0*1	123*888*444	123
G1*2	123*888*444	888*444
G2*1	123*888*444	444
G3*1	123*888*444	(null)
G1*1	*WRITTEN 21 DEC 1984	WRITTEN 21 DEC 1984

Sample Usage of 'G' (GROUP EXTRACTION) Code.

6.43.2 'L' CODE : CORRELATIVE AND CONVERSION LENGTH CODE

The LENGTH code places length constraints on which data values will be returned, or returns the length of the data.

FORMAT:

Ln{,m}

PARAMETER DEFINITIONS:

n	used by itself, is the exact number of characters the defined attribute data value must match. When "m" is present, "n" equals the minimum length parameter, of a length range.
m	when present, equals the maximum data length parameter.

EXAMPLES:

L0	Returns the length of the data string submitted to the length processor.
L7	Return the data value if it is equal to 7 characters long. If the data does not meet the criteria then null is returned.
L3,5	Return the data value if it is greater than or equal to 3 characters long or less than or equal to 5 characters long.

Sample Usage of the 'L'ength Code.

6.43.3 'R' CODE : CORRELATIVE AND CONVERSION RANGE CODE

The RANGE code returns data values which fall within the specified ranges. Multiple ranges are allowed.

FORMAT:

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

PARAMETER DEFINITIONS:

n is the minimum range parameter.

m is the maximum range parameter.

When using negative range sets, the most negative number must be stated first.

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

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

EXAMPLES:

R100,200	Returns the data value if it falls within the ranges of 100 to 200.
R400,600;750,950	Returns the data value if it falls within the ranges of 400 to 600 or 750 to 950.

Sample Usage of the 'R'ange Code.

6.43.4 'P' CODE : CORRELATIVE AND CONVERSION PATTERN CODE

The 'P' code places restrictions on output based on PATTERN matching the value of an attribute.

FORMAT:

P(op){;(op)....}

PARAMETER DEFINITIONS:

P	is the PATTERN MATCH code.
op	is a literal or a pattern match operator.

The Pattern Match code returns data values which match the specified pattern. If the data does not match the specified pattern exactly, then null is returned.

Any combination of PATTERN MATCH operators is allowed.

PATTERN MATCH OPERATORS:

nN	An integer number followed by the letter 'N', which tests for that number of of numeric characters.
NA	An integer number followed by the letter 'A', which tests for that number of alpha characters.
NX	An integer number followed by the letter 'X', which tests for that number of alpha-numeric characters.
"LITERAL"	A literal string, which tests for that literal string.

EXAMPLES:

P(3N-2N-4N);(9N)	Returns Social Security numbers, either 9 numeric characters or 3 numeric, hyphen, 2 numeric, hyphen, 4 numeric.
P((3N) 3N-4N);(10N)	Returns telephone numbers, either 10 numeric characters or 3 numeric in parenthesis, space, 3 numeric, hyphen, 4 numeric.

Sample Usage fo the 'P'attern Code.

6.43.5 'S' CODE : CORRELATIVE AND CONVERSION SUBSTITUTION CODE

The 'S' code substitutes the data value of a referenced attribute with a specified attribute, if the original value is not null or zero.

FORMAT:

S; op1;op2

PARAMETER DEFINITIONS:

S is the SUBSTITUTION code

op1 may either be a delimited text string or an AMC numeric value which is used for the substitution if the data value tested is not null or not zero.

op2 may either be a delimited text string or an AMC numeric value which is used for the substitution if the data value tested is null or zero.

The Substitution code substitutes the current data value with op1 if the data value is not null or zero. If the current data value is null or zero, it will be substituted with op2.

EXAMPLE:

S;4;'XXX'	This specifies that if the data is not equal to zero or null, then it will be replaced by the contents of attribute 4. If it is equal to zero or null, it will be replaced by the string 'XXX'.

An additional feature of the Substitution code is that if it is used in conjunction with the Function Processor, it may serve to test for null or zero, and take different actions according to what kind of data it encounters.

F1(S;*;'NORMAL VALUE')	This specifies that if attribute 1 is null or zero, the string "NORMAL VALUE" will be used, otherwise the contents of attribute 1 will be used.

Sample Usage of the 'S'ubstitution Code.

6.43.6 'C' CODE : CORRELATIVE AND CONVERSION CONCATENATION

The 'C' code provides the facility to concatenate attributes and/or literal values.

FORMAT:

C{x}element x {element x ....}

PARAMETER DEFINITIONS:

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

EXAMPLES:

ATTRIBUTE VALUES	C CODE	RESULTANT OUTPUT
014 SMITH 015 JOHN H.	C;"NAME":14,15	NAME:SMITH,JOHN H.
001 DIME 002 DOZEN	C;1/2	DIME/DOZEN
001 DICK PICK	F;'ATT1=';(C*:1)	ATT1=DICK PICK

Sample Usage of 'C'oncatenate Code.

6.43.7 'T' CODE : CORRELATIVE AND CONVERSION TEXT EXTRACTION

The 'T' code is used to extract a fixed number of characters from an attribute value.

FORMAT:

T{m,}n

PARAMETER DEFINITIONS:

T	is the text extraction code.
m	is the optional starting column number.
,	Is the necessary separator when 'm' is specified.
n	is the number of characters to extract.

A contiguous string of characters may be extracted from an attribute value via the use of a 'T' code. This is useful for fixed field data, or for truncating data, which is sometimes neccessary to prevent folding.

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

If the form 'Tm,n' is specified, then 'n' characters, starting at column 'm', will be extracted. The 'Tm,n' form always counts columns and extracts characters from left to right, regardless of the attribute definition's V/TYPE.

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

Sample Usage of 'T' (Text Extraction) Code.

6.43.8 'D' CODE : CORRELATIVE AND CONVERSION DATE CODE

The 'D' code provides the facility for converting dates to or from a compact internal format suitable for arithmetic processing.

FORMAT:

D{n}{*m}{s}

PARAMETER DEFINITIONS:

D is the DATE FORMAT code..

n is an optional single digit number which specifies the number of digits to occur in the year on output. If 'n' = 0, no year will appear in the date. n = 0, 1, 2, 3 or 4 is valid; n = 4 is assumed default.

* Stands for any single non-numeric character which specifies the delimiter between fields for group extraction. (* May not be a system delimiter.)

m is a single digit number (required parameter if * is specified), that specifies the number of fields to skip for group extraction.

s stands for either any single non-numeric character that may be specified to separate the day, month and year on output, or a special date sub-code. If 's' is specified, the format will be like 12-31-1967. If 's' is not specified, the format will be like 31 DEC 1967.

DATE FORMAT SUB-CODES

D	Day of month.
I	Internal date. Reverse conversion.
J	Julian day of year.
M	Month numeric.
MA	Month alphabetic.
Q	Quarter numeric.
W	Weekday numeric. Monday = 1.
WA	Weekday alphabetic.
Y	Year. Default = 4 digits.

Dates may be stored in items as numbers, and may be printed in any of several different formats in a listing, by use of the 'D' (Date conversion) code. This allows the user to store dates with fewer bytes of disc space, and to perform mathematical calculations involving stored dates.

NOTES:

Notice the use of the '*m' option, which will perform a Group Extraction before applying the date conversion. (See 'G' CODE.)

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

EXAMPLES:

D CODE	INTERNAL FORMAT	EXTERNAL (LISTING) FORMAT	
D	5992	27 MAY 1984	
D/	5992	05/27/1984	
D-	5992	05-27-1984	
D0	5992	27 MAY	
D0/	5992	05/27	
D2*	5992	05*27*84	
D%1	ABC%5992	ABC%27 MAY 1984	
D%1/	ABC%5992	ABC%05/27/1984	
D%1-	ABC%5992	ABC%05-27-1984	
D0%1	ABC%5992	ABC%27 MAY	
D0	ABC%5992	ABC%5992	
D4-	1234	05-18-1971	
DY	1234	1971	(4-digit year)
D2Y	1234	71	(2-digit year)
DQ	1234	2	(2nd quarter)
DD	1234	18	(18th day)
DM	1234	5	(5th month)
DMA	1234	MAY	
DJ	1234	138	(138th day of year)
DW	1234	2	(2nd day of week)
DWA	1234	TUESDAY	
DI	5/18/71	1234	(Reverse conversion)
DI]D2	5/18/71	18 MAY 71	(Multi-value conversion)

Sample Usage of 'D' (Date Conversion) Code.

6.43.8.1 INTERNAL DATE FORMAT

The INTERNAL DATE format allows storage of the date in a form which uses less bytes of storage on the disk and also allows mathematical manipulating of that internally stored value. Zero Date equals December 31, 1967.

NOTES:

The internal format of any date is the integer number of days between that date and the zero date, December 31, 1967. Dates before 12/31/67 are stored as negative numbers; dates after 12/31/67 are stored as positive numbers. To give the user a feel for the use of dates in internal format, examples of dates in both internal and external (listing) format are shown below.

EXAMPLES:

INTERNAL FORMAT	EXTERNAL (LISTING) FORMAT
-173573	12 OCT 1492
-100000	19 MAR 1694
-69942	04 JUL 1776
-10000	14 AUG 1940
-1000	05 APR 1965
-100	22 SEP 1967
-10	21 DEC 1967
-1	30 DEC 1967
0	31 DEC 1967
1	01 JAN 1968
10	10 JAN 1968
100	09 APR 1968
1000	25 SEP 1970
10000	18 MAY 1995
11689	01 JAN 2000
100000	13 OCT 2241

Sample Dates in Internal and External (Listing) Format.

6.43.9 'MT' CODE : CORRELATIVE AND CONVERSION MASK TIME CODE

The MT code provides the facility for converting times to or from a compact internal format suitable for arithmetic processing.

FORMAT:

MT{H}{S}

PARAMETER DEFINITIONS:

MT is the Mask Time code specification.

H is the capital letter H, which optionally specifies 12 hour format. If 'H' is omitted, 24 hour (military) format is assumed.

S is the capital letter S, which optionally specifies seconds on output. If 'S' is omitted, seconds are not listed on output.

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

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

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

MT CODE	INPUT VALUE	STORED VALUE	OUTPUT VALUE
MT	12	43200	12:00
MTH	12	0	12:00AM
MTS	12	43200	12:00:00
MTHS	12	0	12:00:00AM
MT	12:15AM	44100	12:15
MTH	12:15AM	900	12:15AM
MT	1	3600	01:00
MTH	1	3600	01:00AM
MT	6AM	21600	06:00
MTH	6AM	21600	06:00AM
MT	1PM	3600	01:00
MTH	1PM	46800	01:00PM
MT	13	46800	13:00
MTH	13	46800	01:00PM
MT	XYZ	NULL	BLANK

Sample Usage of MT (MASK TIME) Code

6.44 DEFINING FILE TRANSLATION: Tfile CODE

The Tfile code provides a facility for converting a value by translating through a file.

FORMAT:

T{DICT} [file;cn{;vmc};i-amc;o-amc]{;b-amc}

ARGUMENTS:

T is the code name.

file is the name of the file through which translation takes place. The file name preceded by "DICT" indicates a dictionary.

c is the translate sub-code, which must be one of the following:

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

n is an optional value mark count specification. If the c element is followed by a number, the translate will return only the value in VMC n, instead of the complete collection of values concatenated together with blanks. Subvalues will be returned with included blanks.

i-amc is the decimal attribute number for input conversion (in PICK/BASIC). The input value is used as an item-id in the specified file, and the translated value is retrieved from the attribute specified by the i-amc. (If the i-amc is omitted, no input translation takes place.)

o-amc is the attribute mark count for output translation. When ACCESS creates a listing, the attribute values will be looked up in the specified file, and the attribute specified by the o-amc will be listed instead of the original value.

b-amc if specified, will be used instead of o-amc during the listing of break-on and total lines.

The value to be translated is used as an item-id for retrieving an item from the defined translation file. The translated value is retrieved from the specified attribute of the item.

```
Item CARDS in DICT MUNCH is:

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

Data section of MUNCH file is:

Item-id:      DENNY'S              CARLS-JR             MEYERHOFS
    001:      HAMBURGERS           HAMBURGERS           SANDWICHES
    002:      BRISTOL STREET       BRISTOL STREET       S.COAST VILLAGE
    003:      5563072              9792231              8830002
    004:      MC]V                 NONE                 MC]V]BA

Data section of CARD-FILE is:

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

>LIST MUNCH "DENNY'S""CARLS-JR""MEYERHOFS" CARDS HDR-SUPP [CR]

MUNCH..... CREDIT CARDS.......

DENNY'S     MASTER CHARGE
            VISA
CARLS-JR    NONE
MEYERHOFS   MASTER CHARGE
            VISA
            BANKAMERICARD
```

Sample Usage of the Tfile (Translate) Code.

6.45 DEFINING ASCII AND USER CONVERSIONS: MX AND U CODES

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

FORMAT:

MX

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

FORMAT:

Unxxx

The 'U' code specifies an entry point into a user-written piece of software.

where:

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

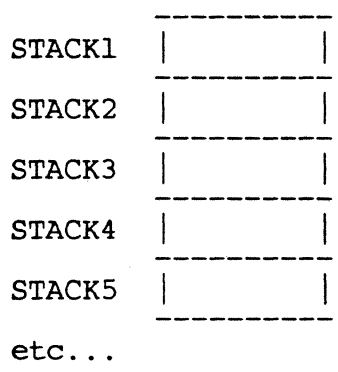
WARNING: Do not use the U code unless you fully understand its action at the assembly-code level!

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

Sample Usage of the MX (Mask Hexadecimal) Code.

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

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



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

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

FORMAT:

Felement;element;element...

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

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

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

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

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

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

'F' Code Operators.

```
F Code        F;C3;C2;C1;+;*

STACK1  3     STACK1  2       STACK1  1     STACK1  3     STACK1  9
STACK2        STACK2  3       STACK2  2     STACK2  3     STACK2
STACK3        STACK3          STACK3  3     STACK3        STACK3
STACK4        STACK4          STACK4        STACK4        STACK4
STACK5        STACK5          STACK5        STACK5        STACK5
```

Sample 'F' Code and Associated Operations on Stack.

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

(attribute3 * attribute4) + attribute2

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

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

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

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

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

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

Examples of Function codes.

6.47 F CODE SPECIAL OPERANDS

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

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

Any conversion may be specified in the body of a Function correlative. The conversion specification must be enclosed by parentheses.

Special 2-character operands may be used as 'F' code elements, as listed in the table of operands below. For example:

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

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

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

F;10;11;(TDICT SALES;X;3;3);*	Places the data from attribute 10 in the stack; translates the data from attribute 11 through the file named SALES and stacks it; then multiplies the two numbers together.
F;D;(DY);3;(DY);-	Computes the difference in years between the current date and the date in attribute 3.
F;1;(ML#10);2;:	Concatenates the data in attribute 2 with the result of applying the format string "L#10" to attribute 1.

Sample Usage of F-Correlatives with Conversions.

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

F-Code Counter Operands.

6.47.1 The Load Previous Value (LPV) operator.

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

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

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

002 5                                   Data attribute 5.
    .
    .
008 G*1]MR%8]F;LPV;"52";R;"*C";:]TFILE;C;;3

This has rather less motivation, since it is equivalent to

008 F;5(G*1]MR%8);"52";R;"*C";:;(TFILE;C;;3).
```

Use of the LPV operator in F-correlatives.

The LPV should be used as the first operator in an F-correlative, because it has the effect of loading the contents of the temporary data area into the stack. If the LPV is used at other points in an F-correlative, strange things will happen.

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

6.48 SUMMARY OF F CODE STACK OPERATIONS

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

OPERAND	ACTION
n{R}	(a decimal number) The attribute value for the corresponding attribute number is stacked.
Cn	(where 'n' is a decimal number) The integer constant 'n' is stacked.
"String"	The literal string enclosed in double quotes is stacked.
'String'	The literal string enclosed in single quotes is stacked.
D	The current date (in internal format) is stacked.
T	The current time (in internal format) is stacked.
P	The top stack element is pushed back onto the stack.
NI	The current item counter is stacked.
ND	The number of detail lines since the last control break is stacked. (This number is 1 on detail lines, and is the same as the item counter on grand-total lines.)
NS	The current sub-multi-value counter is stacked.
NB	The current control-break number is stacked.

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

[]	The sub-string from the string in the third stack element is extracted, starting from the character position defined in the second stack element, and the number of characters defined in the top stack element.

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

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

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

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

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

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

6.49 DEFINING MATHEMATICAL FUNCTIONS: THE A CORRELATIVE

The A-code is designed to perform the same function as the F-code, but it is writtem in a format which is both simpler to write and easier to understand than the format of the F-code.

FORMAT:

A(expression)

Where an expression is made up of operands, functions and operators, as described below.

OPERANDS

AMC NUMBERS

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

AMC NAMES

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

The operation of the "N" function is as follows:

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

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

Note that an A-code can now call another A-code by name, and that the second A-code can specify a third A-code, and so on. However, no attempt is made to assure that an A-code does not call itself. If this is attempted, or any time that "nested" calls are made more than seven (7) levels deep, the ACCESS compiler will abort with a RTN STACK FULL message.

LITERAL NUMBERS

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

LITERAL STRINGS

Any literal string, enclosed in single quotes (') or double quotes (") is a legal operand.

SPECIAL OPERANDS

The A-code has several special system operands which are the same as for F-codes. They consist of:

NI -- the item counter

NV -- the value counter

NS -- the sub-value counter

ND -- the detail-line counter

NB -- the break level counter

LPV -- load previous value

D -- the system date (in internal format)

T -- the system time (in internal format)

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

FUNCTIONS

REMAINDER FUNCTION: "R"

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

SUMMATION FUNCTION: "S"

The summation function takes one expression as an operand, and works the same way as the S operator in the F-codes. For example, S(4) will sum any multi-values of attribute 4.

The summation operator may appear anywhere in an A-code.

SUB-STRING FUNCTION

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

ARITHMETIC OPERATORS

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

RELATIONAL OPERATORS

The relational operators <, >, >=, <=, = and # denote the logica relations greater-than, less-than, greater-than or equal to, less-than or equal to, equal and not-equal respectively. Each of the relational operators takes two expressions as operands, and evaluates to 1 (true) or 0 (false) depending whether or not the indicated relation holds between the two operands. For instance, "1">="2" evaluates to 0 (false) because the number 1 is not greater-than or equal to the number 2. Expressions involving these and other A-code operators are written much like DATA/BASIC expressions.

NOTE: The precedence of the operators is important to keep in mind when writing an A-code. In the absence of parentheses to indicate the order in which operators are to be applied, multiplications and divisions have greater precedence than addition and subtraction which in turn have greater precedence than the relational operators. If two operators have the same precedence, they are applied from left to right. For example, 1*2+3<4 will evaluate as ((1*2)+3)<4, but 1>=2-3/4 will evaluate as 1>=(2-(3/4)). Also, 1+2-3 will evaluate as (1+2)-3, and 4/5*6 will evaluate as (4/5)*6. 20 Levels of parentheses nesting are allowed in A-codes.

A-CODE	MEANING
A1+2	Adds attributes 1 and 2.
A"10"*3	Multiplies the value of attribute 3 by 10.
AS(4+"25")	Adds 25 to each value of attribute 4, then sums the multi-values.
AN(INV-AMT)-N(BAL.DUE)	Subtracts the value of the attribute defined by BAL.DUE in the dictionary from the value of the attribute defined by INV-AMT.
AN(SS-NUM)['4','2']	Returns the 4th and 5th digits of the attribute specified by SS-NUM.

Sample usage of the Mathematical 'A' function.

6.50 HANDLING NUMBERS AND FORMATTING: MR AND ML CODES

The MR and ML codes allow special processing for numbers or dollar amounts, and allow special formatting.

The ACCESS MR and ML codes function exactly the same as the 'R' and 'L' format strings in PICK/BASIC.

FORMAT:

M(R/L){n{m}}{Z}{,}{C/D/M/E/N}{$}{ (format-string) }

where:

M	is the code name. MR specifies the number to be right justified; ML specifies left justification.
n	is a single decimal digit (0-9) which specifies the number of digits to be printed to the right of the decimal point. If 'n' is not specified, 0 is assumed. If 0 is assumed or specified, no decimal point will be printed.
m	is a single digit number (0-9) which specifies that the number on file is to be 'descaled' (divided) by that power of ten. (That is, if m=2, the number is divided by 100, if m=3, the number is divided by 1000, and so on.) The number 'm' is the number of implied digits to the right of the decimal point for the number as it is stored in the file. If m > n, then the number will be rounded off, either up or down, to 'n' digits.
Z	is the optional zero-suppress parameter. If 'Z' is specified, the number 0 (zero) will be printed as blanks.
,	specifies insertion of commas every three digits to the left of the decimal place.
C	causes negative values to be followed by the letters CR.
D	causes positive values to be followed by the letters DB.
M	causes negative numbers to be followed by a minus sign (-).
E	causes negative numbers to be enclosed inside angle brackets (< and >).
N	causes the minus sign on negative numbers to be suppressed.
$	appends a dollar sign ($) to the number before justification.

The format mask specification, which is enclosed in parentheses, consists of format codes and literal data. A format code is one of the characters #, *, or %, optionally followed by a number to indicate that number of repititions of the character.

#n	specifies data to be justified in a field of 'n' blanks.
*n	specifies data to be justified in a field of 'n' asterisks (*).
%n	specifies justification in a field of 'n' zeroes (0).

Any amount of alphabetic data may also be specified inside the parentheses in the format mask specification. The data will be printed exactly as specified in the format mask, with the number being processed appearing either right or left justified in the place of the #'s, *'s or %'s.

NOTE: the $ option appends a dollar sign to the number, and then justifies the number, so the dollar sign will always appear just before the first digit of the number on output.

DATA	CONVERSION	RESULT
1234	MR2	12.34
1234	MR	1234
1234	MR(%10)	0000001234
1234	MR(*10)	******1234
12345678	MR2,E	123,456.78
-12345678	MR2,E	<123,456.78>
-12345678	MR2,C$	$123,456.78CR
572082394	MR24	57208.24 (Note the Rounding)
572082394	MR2,$(#20)	$5,720,823.94
572082394	MR2,($#20)	$ 5,720,823.94
572082394	MR2,$(*20)	*******$5,720,823.94
572082394	ML(###-##-####)	572-08-2394
572082394	MR(#3-#2-#4)	572-08-2394
572082394	ML(#3-#4 EXT.#2)	572-0823 EXT.94

Sample Usage of the MR and ML Codes.

6.51 ADDITIONAL CHARACTER MANIPULATION: MC CODE

The MC (Mask Character) Code allows the user to change attribute data to upper or lower case, or to select out certain classes of characters.

Following is a list of the legal forms of the MC code and their function:

MCU — Converts data to upper case. Will change all lower-case letters to upper case; has no effect on upper-case letters or non-alphabetic characters.

MCL — Converts data to lower case. Will change all upper-case letters to lower case; has no effect on lower-case letters of non-alphabetic characters.

MCA — Extracts and prints all Alphabetic characters, either upper-case or lower-case; non-alphabetic characters will be deleted from the data.

MCN — Extracts all numeric characters (0-9) from the data; deletes all other characters.

MC/A — Extracts all non-alphabetic characters from the data; deletes all alphabetic characters.

MC/N — Extracts all non-numeric characters; deletes all numeric characters.

VALUE	MC CODE	OUTPUT VALUE
JOHN SMITH 1234	MCL	john smith 1234
John Smith 1234	MCU	JOHN SMITH 1234
John Smith 1234	MCA	JohnSmith
John Smith 1234	MCN	1234
572-08-2394	MCN	572082394
(714) 552-4275	MCN	7145524275
abc123$%XYZ	MC/A	123$%&
abc123$%XYZ	MC/N	abc$%XYZ

Sample Usage of the MC (Mask Character) Code.

6.52 SPECIAL CONTROL CHARACTERS

This appendix describes special control characters which can be used during input to the user's terminal.

CONTROL CHARACTER	FUNCTION
Control-H	Backspace: Deletes last character typed and prints the backspace character as defined by the TERM verb.
Control-X	Cancel: deletes entire line and prints a carriage return. Will cause a listing to terminate if it is being sent to the screen, and it is pausing at the end of the page.
Control-R	Retype: causes the entire current line to be reprinted on the next line. This is useful after backspace characters have been typed.
Control-W	Backup one word: backspaces until a character which is neither a number nor a letter is reached.
Control-Shift-O	Line Continuation character: if typed as the last character on a line (before the carriage return) will allow the line being typed to be more than 140 characters long. Additional lines will be prompted for with a colon (:) prompt character.

SECTION 7

ICON/PICK PERIPHERALS

ICON®

Chapter 7

PERIPHERALS

THE PICK SYSTEM

USER MANUAL

7.1 AN OVERVIEW

The PICK system has two classes of peripheral devices: one magnetic tape drive and one or more printers. It is the nature of each of these devices that each can service only one user at a time. This section discusses the facilities in the PICK system which allow the convenient use of these devices in a multi-user environment.

The tape drive and printer do different things and as such are controlled by different subsystems. The tape drive is both an input device and an output device, and in both modes can transfer either machine-readable (data and programs) or people-readable (text) data. It may be used by only one process at a time, and that process must be a user process with a terminal attached or an equivalent phantom process. The routines available for tape manipulation include the ability to acquire and release the tape drive and to transfer data to and from data files and print files.

The function of a printer is simpler, and as such has a more extensive control subsystem called the spooler, which separates the generation of output to printers from the actual printing process. It allows many processes to generate output to a printer at the same time without conflict over the physical device, and it allows flexibilty as to what gets printed where and when.

The process of generating a report and obtaining it from the printer is as follows. The user process executes a routine which generates print output which is stored on disk and creates a control record pointing to the print file. Another process, known as the spooler, finds the print file and executes the actual printing process.

The spooler runs as a separate process which is numerically the last process on the system. Since the terminal for this line does not exist, this is often referred to as a "phantom process". The spooler is automatically started by either a "FILE-RESTORE" or a "COLD-START" and is automatically assigned by the system to the last hardware terminal communication line in the system + 1. The spooler may also be restarted by use of the :STARTSPOOLER verb.

The following is an overview of features of the spooler. Precise discussion of each is to be found in the relevant sections following. The verb names have been included where relevant for quick reference.

SPOOLER SYSTEM FEATURES

- GENERAL STRUCTURE.

  - Six hundred print files may be retained and controlled.

  - Sixty print files may be generated simultaneously.

  - One hundred twenty-five different output queues are available

  - Up to 125 copies of a report may be requested as one print job.

  - Up to twenty printers may be defined -- 4 parallel, 16 serial.

  - Print files are printed on four parallel printers concurrently by one process.

  - Up to sixteen ports may be used as serial printer spoolers.

  - Print files may be transferred to and from tape.

  - Print files are transferred to and from tape by the requesting process, not by the spooler.

- PRINT FILE DEFINITION FEATURES. (SP-ASSIGN)

  - Control of print file destination -- tape, printer or hold file.

  - Control of time of enquement -- immediate, at completion or from a hold file.

  - Control of time of print file deletion -- concurrent with printing, at completion of printing or from a hold file.

  - Control of the use of disc storage by the print file.

  - Specification of the number of copies to print, up to 125.

  - Specification of one of 125 output queues.

  - Predefinition of several print files generated by a single job, such that each has its own assignment specification.

  - Print files may be held open across several jobs.

  - Open print files may be closed when desired.

  - Open print files will be automatically closed when necessary, if unintentionally left open.

  - The current specification of the line may be displayed.

- PRINT FILE CONTROL FEATURES. (SP-EDIT)

  - Print file security: Print files may be inspected and printed only by the generating account, or by the system manager.

  - Ease of print file identification.

    - All print file entry numbers are displayed at print file generation time.

    - Only the account's print files are returned.

  - Numerically contiguous blocks of print files may be accessed, subject to ownership considerations.

  - Print files specified to be enqueued to a particular output queue, or a numerically contiguous group of output queues, may be accessed, subject to ownership considerations.

  - Print files not being output may be disenqueued. Non-permanent files will become hold files. (SP-KILL)

  - The first 500 bytes of each print file available to the user may be displayed.

  - A text string may be specified as the address of the line which is to be the start of the output text.

  - Destination, printer or tape, may be specified by options on the SP-EDIT verb, without reassigning the line.

  - Selected groups of print files may be spooled or deleted without using the prompt sequence.

  - The output queue specification of a print file may be changed.

  - The copy count specification of a print file may be changed.

  - A print file may be sent to the user's terminal a page at a time, or continuously.

  - In page at a time mode the user may go back a page, go back to the top, or exit from the end of each page.

  - A print file may be sent to a file as a set of items in RUNOFF format.

  - Each print file may be deleted when desired.

  - The system manager may inspect, print, and delete any print file.

- The system manager may access print files by the name of the account which generated them.

- The system manager may inspect the first 500 bytes of any enqueued print file.

- The system manager may not send a print file to a file in RUNOFF format unless he is logged on to the generating account.

- PRINTER CONTROL FEATURES.

  - From one to twenty printers may be declared. (STARTPTR)

  - Each has a unique printer ordinal, device specification, output queue list, and inter-job page eject count.

  - Each printer can output the contents of one, two, or three output queues.

  - Any number of printers can output the contents of a given output queue at the same time.

  - Print files may be aligned on each printer. (STATRPTR)

  - The output on each printer can be terminated immediately. (SP-KILL)

  - Print file output may be terminated while the print file is being generated.

  - Serial printers will normally paginate on the basis of a page-eject (X'0C') character.

  - Serial printers which do not recognize a page-eject character can be informed of the fact, and informed of the number of lines per page to print.

  - Serial printers may caused to suppress the initial page-eject in each print file.

  - Each printer can be set to stop at the completion of the current job. (STOPPTR)

  - Each printer may be deleted from the spooler system. (SP-KILL)

- SPOOLER SYSTEM CONTROL FEATURES.

  - Print file entry numbers are returned to the PROC secondary input buffer and marked as entry numbers.

  - The print file control block record may be displayed. (LISTPEQS)

- The print file control block record contains:
    - The entry number of each print file.
    - The status of each print file.
    - The next link in the output queue for each print file, as applicable.
    - The output queue specified for each print file.
    - The number of copies requested for each print file.
    - The size in frames of each print file.
    - The date and time of generation of each print file.
    - The name of the account which generated the print file.
    - The total number of frames used by the listed control block records.
- The print file control block can display:
    - All control block records, including records of deleted print files.
    - All current control block records.
    - All print files generated by a named account, either current, or current and deleted.
    - The chain of print files enqueued in each output queue by queue, by print file sequence in the queue.
    - Any group of numerically contiguous entries or queues, and with the above conditionals.
    - Just the total number of frames referenced in any of the above cases.
- The printer control block may be printed. (LISTPTR)
- The printer control block display includes:
    - The printer ordinal (identification number).
    - The printer type -- serial or parallel.
    - The output queues to which it is allocated.
    - The inter-job page eject specified for the printer.
    - The parallel printer number or line number.
    - The current status of the printer.

- The data from the LISTPTR display is inserted in the PROC secondary input buffer in order to allow PROC control of the printer system.

- The assignment specifications of all lines on the system may be displayed. (LISTABS)
- The status of the spooler system may be displayed. (SP-STATUS)

    - The status of all printers, one printer, or a numerically contiguous printers may be selected for display.

- The SP-STATUS verb leaves error message numbers indicating the condition of the spooler system in the PROC secondary input buffer.

- The COLD-START routine leaves print file control record area intact, if possible.

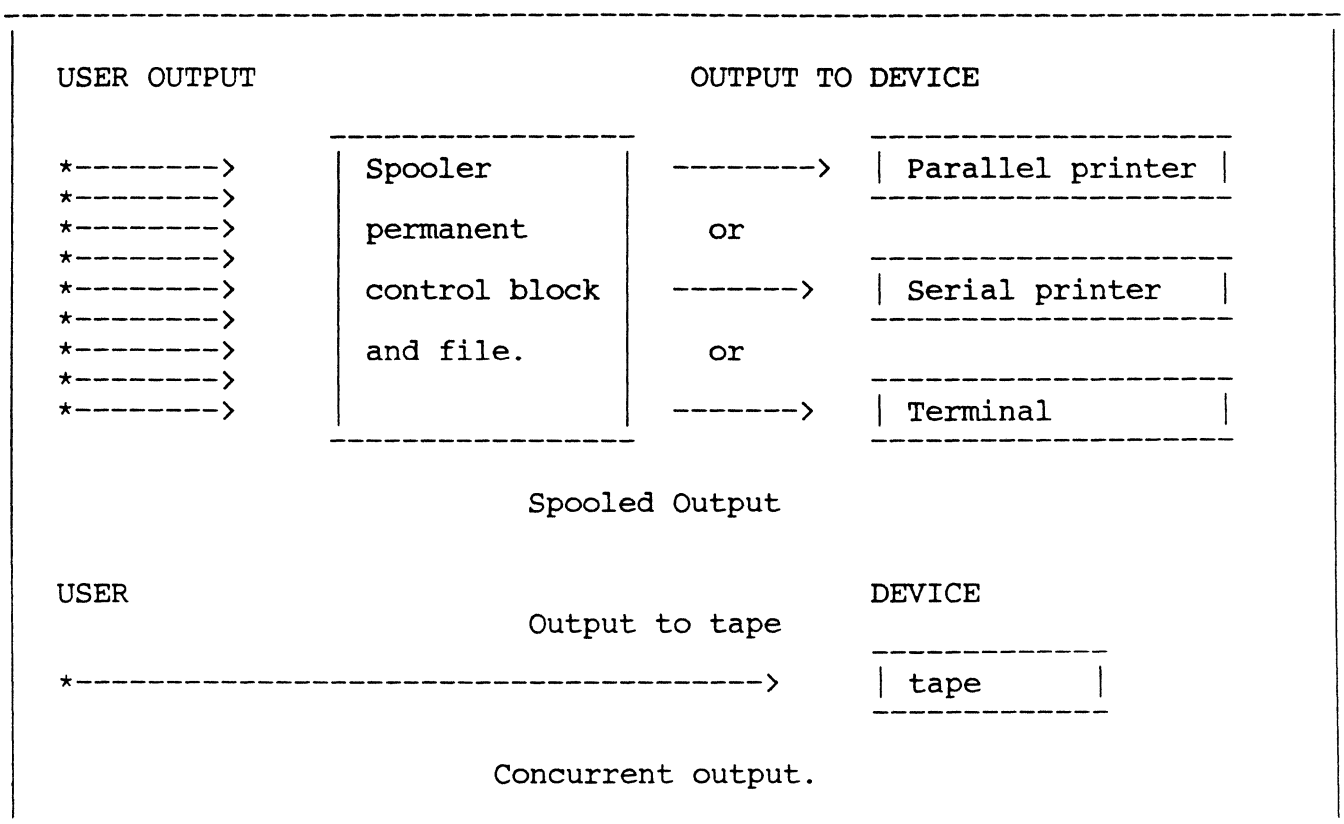
- :STARTSPOOLER allows the following levels of reinitialization:

    - Reinitialization of the major control pointers.

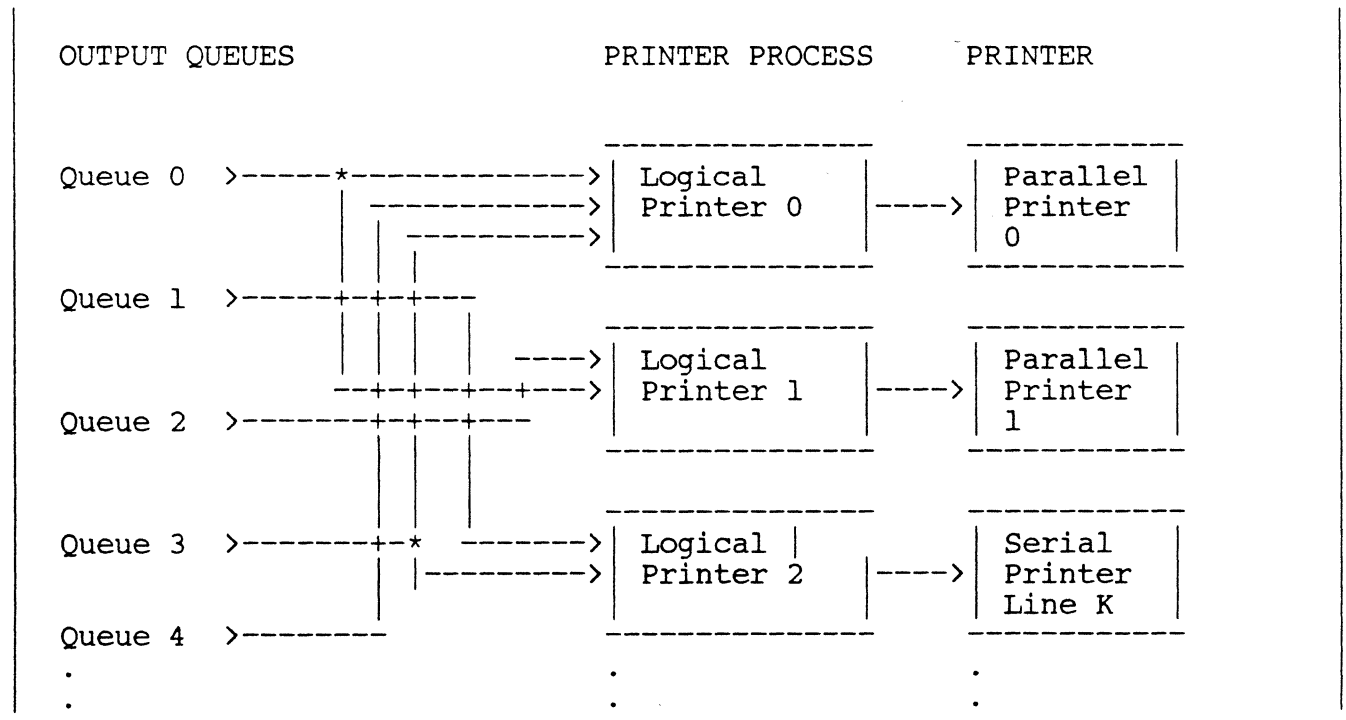
    - Reinitialization of all printers, input queues and output queues, excepting the print file control records, and includes a inspection or print file control record for admissability.

    - Reinitialization of the whole spooler workspace and table area, as on a file restore.

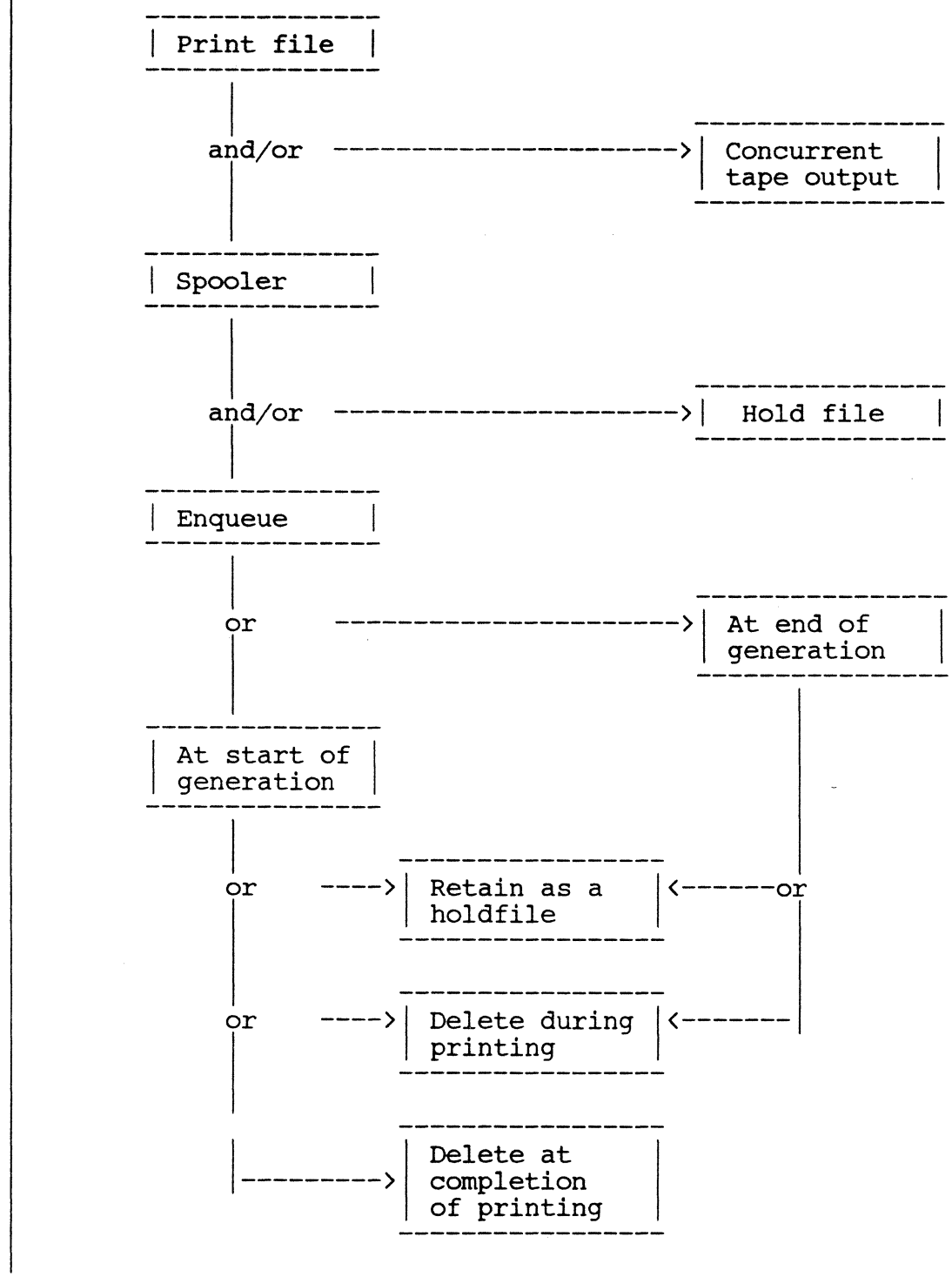
    - The linking-up of work space for all other lines on the system which are not logged on.



The Spooler



A case of queue to printer bindings.



The destinations of a print file.

Since the spooler is a unique software process, a complete set of TCL verbs is supplied to communicate with it. These verbs are summarized in Figure A. The remaining topics within this chapter present detailed descriptions of each verb.

OUTPUT SPECIFICATION

Pick allows the user to specify the destination of the output of each job by the use of the SP-ASSIGN verb. Default assignment is set up at logon time, and the verb only needs to be used if the current assignment needs to be changed. Output may be sent to the printer or tape drive and/or retained as a hold file, or suppressed all together.

PRINT FILE CONTROL

The use of the SP-EDIT, SP-ASSIGN, and SP-KILL verbs allows considerable flexibility over the disposition of print files. The key to control is the hold file, which is simply a print file which has been specified by the SP-ASSIGN verb to be retained, or which has been disenqueued by the SP-KILL verb, and will be retained as a result. These hold files are the object of the SP-EDIT verb. In association with the SP-ASSIGN verb, the SP-EDIT verb allows enqueuement of hold files to any desired output queue, and thereby to the desired printer, modification of the number of copies requested, display of the print file to a CRT or the printing of the print file on a printer auxiliary to a CRT, translation of the print file into a sequence of file items in RUNOFF format, or deletion.

HOLD FILES

The creation of hold files permits reports to be spooled to the disc and held for an indeterminate length of time for subsequent or multiple output. The spooler structure will accommodate a maximum of 600 print files. Print files are normally created as hold files in any case when recreation of the print file might be inconvenient or time consuming. They are particularly recommended if the printer tends to jam or the communication line to a remote printer is tempermental, if the process generating the print file also updates the relevant data records, if the decision at to when and where to print the report has not been made at generation time, or if there is a reasonable possibility that more copies of the report may be desired. The drawbacks of hold files are that they require considerable amounts of disc space, that they must be explicitly deleted, and that they are not saved by the formatted file save process. If a restore is done from a file-save tape, the hold files will disappear. If necessary, they may be spooled to tape for more secure retention, and restored via the SP-TAPEOUT verb. They will remain after a COLD-START if the print file control records pass certain admissability tests.

PRINTER CONTROL

Control of the multiple printers on the system uses the STARTPTR, STOPPTR, and SP-KILL verbs. The STARTPTR verb associates the physical printer and its associated process with the desired output queues. It also defines the printer ordinal by which the printer is referenced, the number of pages to skip between jobs, the type of printer involved, and executes alignment when desired. The STOPPTR verb indicates that the printer process is to stop at the completion of the current job, and is required prior to the execution of the STARTPTR verb if the printer has been previously initialized. The SP-KILL verb terminates the execution of the current print job on the specified printer. The SP-KILL verb is also used to delete a printer from the spooler system when the proper option is used.

SPOOLER SUBSYSTEM INTERROGATION

The condition of the various parts of the Spooler subsystem may be interrogated by the use of the LISTPEQS, LISTPTR, LISTABS, and SP-STATUS verbs. The LISTPEQS verb displays the print file control block records in various ways according to the specified options, and allows the user and the system manager to find out what print files are currently resident on disc, their condition, and who generated them when. The LISTPTR verb displays the printer allocations. The LISTABS verb displays the assignments of all the lines on the system. The SP-STATUS verb and thereby the WHAT verb display the condition of the spooler and the definition and current task of each printer.

SPOOLER INITIALIZATION

The Spooler is automatically initialized by the execution of a file restore. The Spooler is reinitialized by the execution of a COLD-START; hold files are retained in this case. Should intialization or reinitialization of the Spooler be desired, the :STARTSPOOLER verb is available. Be aware that much storage can be lost, possibly requiring a restore at an inconvenient time.

DEVICE ATTACHMENT

Only one process at a time may use the tape. The system software keeps track of which process is associated with the tape drive. Users of the printers need not be concerned with attachment/detachment since the spooler is the only process which can use the parallel printers, and only the processes attached to the serial printers can output to the serial printers, under the control of the spooler process.

TAPE MANIPULATION

There are a collection of verbs involved in tape manipulation. They include T-ATT and T-DET to attach the tape drive to a process and then to detach it. T-FWD, T-BCK, and T-REW are available for tape positioning. T-RDLBL and T-READ are available to interrogate the tape. SP-TAPEOUT is available to move print files from tape to print files. T-DUMP and T-LOAD are ENGLISH verbs available to move data items to and from tape. All other verbs in the system which generate print files may send those print files directly to tape, and SP-EDIT may send stored print files to tape. The various verbs involved in the save and restore processes utilize the

tape. Finally, note the tape manipulation operators in BASIC.

VERB	BRIEF DESCRIPTION
LISTPEQS	Displays the print file control records.
LISTPTR	Displays the printer control block.
LISTABS	Displays the assignment of each line on the system.
SP-ASSIGN	Defines the print file destination.
SP-CLOSE	Terminates the SP-OPEN condition, so that the print file is closed.
SP-EDIT	Allows operation on hold files.
SP-KILL	Cancels the current output from a printer, disenqueues a print file, or deletes a printer from the system.
SP-OPEN	Causes a sequence of jobs to be taken as one job for purposes of output.
SP-STATUS	Displays the current status of the Spooler and each defined printer.
SP-TAPEOUT	Prints a tape created by the spooler.
STARTPTR	Initializes a printer.
STOPPTR	Halts printer after current job.
T-ATT	Attaches the tape drive to a line.
T-DET	Detaches the tape drive from a line.

Summary of Spooler Verbs

VERB	BRIEF DESCRIPTION
T-ATT	Attaches the tape drive to a line.
T-DET	Detaches the tape drive from a line.
T-FWD	Moves the tape forward.
T-BCK	Moves the tape backward.
T-SPACE	Causes multiple T-FWDs.
T-EOD	Moves the tape forward to end-of-device.
T-REW	Rewinds -- moves the tape backward to end-of-device.
T-WEOF	Writes an end-of-file mark on the tape.
T-CHK	Checks the tape for parity errors.
T-LOAD	Moves data from tape to disc files.
T-DUMP	Moves data from disc files to tape.
T-READ	Allows inspection of the contents of a tape.
T-RDLBL	Allows inspection of the tape label.
S-DUMP	Sorts data, then moves the data from a disc file to tape.

Tape Verbs.

Device assignments are individually set by each terminal user. Each user can modify his device assignment by use of the SP-ASSIGN verb. The logon process assigns the line printer as the standard output device. This can be altered at any time via the SP-ASSIGN verb.

7.3.1 OVERVIEW OF SP-ASSIGN OPTIONS.

The SP-ASSIGN verb allows the specification of destination of a print file. The H, I, and S specifications are as before. The T specification specifies that the print file shall go directly to tape under the control of the generating process, rather than under the control of the spooler process. Therefore, mount your tape before you start a tape output job. The N specification no longer exists because the spooler is the only process which has access to the parallel printers and tape assignment automatically allocates the tape drive to the process.

The SP-ASSIGN verb has the following specifications. A ? in the options string will display the resulting setting. A ? by itself will display the current setting without disturbing it.

Since thespooler has 126 different output queues (0-125), SP-ASSIGN allows specification of the form queue by use of Fn where n is between 0 and 125, inclusive. There may be no spaces between F and n. The default form number is 0.

A number n in the string not immediately following F will specify the number of copies to print, where n is between 0 and 125, inclusive. The first number n in the options string will be taken to be the copy count, and its successors, if any, will be ignored. The default copy count is 1; and the default assignment is print at completion. These are generated at logon and by executing the SP-ASSIGN verb with no options.

The R specification will open a print file with the specifications given by the other options in the string, and will aviod the automatic print-file closure embedded in the SP-ASSIGN verb. This uses the form Rn, where n is between 0 and 125 inclusive, is concatenated to the R, and specifies the print file name to be used in conjunction with PRINT ON n in basic programs.

The O specification causes the print file to remain open at completion of the process. The C specification means that the process is to be choked. It is allowable only with a I specification and with no H specification. It causes the process to stop generating output when it gets 20 frames ahead of the consuming process. Be aware that if a printer is not available or assigned to other form queues, your process will wait indefinitely if this option is invoked.

The default is: Print one copy on output queue 0 at completion of generation.

H	Holdfile
S	No printed output -- do not link on
I	Link on at initiation
T	Send to tape -- generating process does this
C	Choke the process -- slow generation to output rate
O	Keep the print file open at close time.
Rn	Initiate a print file with this SP-ASSIGN to be generated by PRINT ON n, where n is between 0 and 125, inclusive.
?	Display the curent or resulting assignment.
number	Number of copies, between 1 and 125, inclusive.
Fn	Form number (output queue number) between 0 and 125, inclusive.

OPTION SUMMARY

7.3.2 CLASSES OF SP-ASSIGNMENT PARAMETERS

The various characteristics which can be attributed to a print file are discussed below. Destination specifications define general dispostion of the print file, whether it goes to tape, the printer, a hold file, or nowhere. If it is going to a printer, the destination specifications also define when it is to be enqueued and when its storage is to be released. The form number specifies the output queue into which a print file is to be inserted, and thereby the printer which shall print it. The copy count specifies how many copies of the report shall be generated. Print file predefinition allows different assignments for print files generated simultaneously.

7.3.2.1 Destination specification:

There are four output possible destinations: printer, tape, hold file, and nowhere. Output to the printer is the default. Therefore, the possibilities are:

Not printer	S	Causes output to be not printed.
Tape	T	Causes output to be sent to the tape.
Holdfile	H	Causes output to be retained in a hold file.

Note that the T specification will cause the process to send its print file output directly to tape. This allows control over the tape record size and supression of the tape label. It also means that several different print files created by the PRINT ON statement in basic can be intermixed on the tape. See the discussion of print file predefinition below.

The specifiers used with SP-ASSIGN may be in any order. The above three specifiers may be used in the following combinations with the following meanings.

S	The print file disappears.
H	The print file will become a hold file on disc, and it will be enqueued for output.
HS	The print file will become a hold file. It will not be enqueued for output.
T	The print file will be output to tape concurrently with its generation, and a disc print file will be created which will be enqueued for output.
TS	The print file will be output to tape concurrently with its generation; it will not become a disc print file.
THS	The print file will be output to tape concurrently with its generation; a hold file copy of it will be created on disc, and it will not be enqueued for output.

Compound destination parameters.

The Immediate specification.

There are two enquing timings, immediate and at the end of the job. At the end of the job is the default; therefore:

Immediate	I	Links the job on at the start.

The I flag will only be set in conjunction with output to the printer. It will have no effect on the H option. Tape output is done by the generating process concurrent with the generation; therefore this option has no meaning when attached to a T option. An S option will nullify an I option.

There are four output storage protocols for the single copy case:

1) Hold files are not released to overflow by the spooler. They are released by a specific command within the SP-EDIT processor by the user's process.

2) Closed non-hold files are released frame by frame during output.

3) Open non-hold files are released upon completion of output.

4) Choked open non-hold files are released to overflow frame by frame during output, and the frame counter is decremented.

The difference in release timing allows the SP-KILLing of an open print file without loss of the print file or its associated storage.

The case for multiple copies varies in that the storage can not be released until the last copy. Therefore the following considerations hold: The first copy may be SP-KILLed while it is still open, but it will turn into a hold file. All succeding copies are closed. Storage will be released during the printing of the last copy if release is specified. The multiple copy option is not valid with the choke option, and the specifification of the choke option will cause the copy count to go to 1.

THE CHOKED SPECIFICATION.

Defining a print file as choked, the C option, causes the input processor to enter a wait loop when it is more than 20 frames ahead of the output processor. It will resume when the output processor releases another frame to overflow.

Choke	C	Limits the storage used by a print file.

This option will only be set if immediate ouput to the printer is specified. It has no effect on output to tape, which is inherently synchronous with the generation of output. This option is not available if the print file is a hold file. It is irrelevant to tape output. It operates only in conjunction with the I option. It causes the copy count to be set to one. It will not work if there is not a printer available to process the form number specified for the output.

OPEN SPECIFICATION.

It is possible to keep a print file open at the end of a print generation run, that is, as the user's process goes to TCL or its proc equivalent. This option is compatable with all other options, and has the primary value of keeping a related set of reports together.

Keep Open	O	This flags the print file to be not closed at close time.

Executing the SP-ASSIGN verb will have the effect of closing these files, unless the R option is used in the current SP-ASSIGNment. SP-CLOSE and logging off will force the print files to close.

7.3.2.2 THE FORM NUMBER

The form number specified for a printfile allows ease of printer allocation to print jobs and classes of print jobs. The default form number is 0. The maximum is 125. The format of the form option requires that the form number be concatenated with the F option specifier. Note that "form" is a synonym for "output queue".

Form number	Fn	Where n is the desired form number, where n may be between 0 and 125 inclusive.

This option is irrelevant for tape output. It is not effective until the job is enqueued for output. The job is then enqueued in output queue n. A print job may be enqueued in only one output queue, while each printer may service up to three output queues, and any, some, or all of the printers on the system may service one output queue. The output queue specification for a printfile may be changed by the SP-EDIT function.

7.3.2.3 THE COPY COUNT

The number of copies of a printfile to be output may be specified. The default is one; the maximum is 125. The copy count generated is the first numerical string in the SP-ASSIGNment parameter list not preceeded by an F or an R.

Copy count	n	n is the number of copies to output. n may be between 1 and 125 inclusive.

This option is irrelevant for tape output. It is not useable with the C option. No storage is released until the last copy is being output, in those cases which specify release of storage by the spooler. The copy count specification for a printfile may be changed by the SP-EDIT function. At the completion of output, the copy count parameter will always be decremented to one.

7.3.2.4 Finding out what your assignment specification is.

You can interrogate your assignment specification by using the ?.

Your assignment	?	will output your line's assignment specification

This option may be used as the only option on the SP-ASSIGN verb at which time the assignment will not be modified; but it will be displayed. If the ? option is used with other options, the other options will be inserted in your line's assignment block, and then the contents of that block will be displayed, as in all the examples below.

Note that the LISTABS verb will give the assignments for all lines, and that the discussion of the LISTABS verb includes a list of the indicators and their meanings. The only difference is that when "S" is not

specified, "P" is displayed; and when "S" is specified, "P" is not displayed.

7.3.2.5 PRINTFILE PREDEFINITION

It is possible to preassign the various print files which may be generated by a basic program using the PRINT ON n statement. Execution of an SP-ASSIGN with this option will reserve input and permanent control blocks and obtain the first frame of storage for the printfile. If the printfile is defined as a hold file, the entry number will be issued at this time. If it is to go to tape, the tape will be attached at this time, if it has not already been attached. The format of this option requires that the PRINT ON number of this printfile be concatenated with the R option specifier.

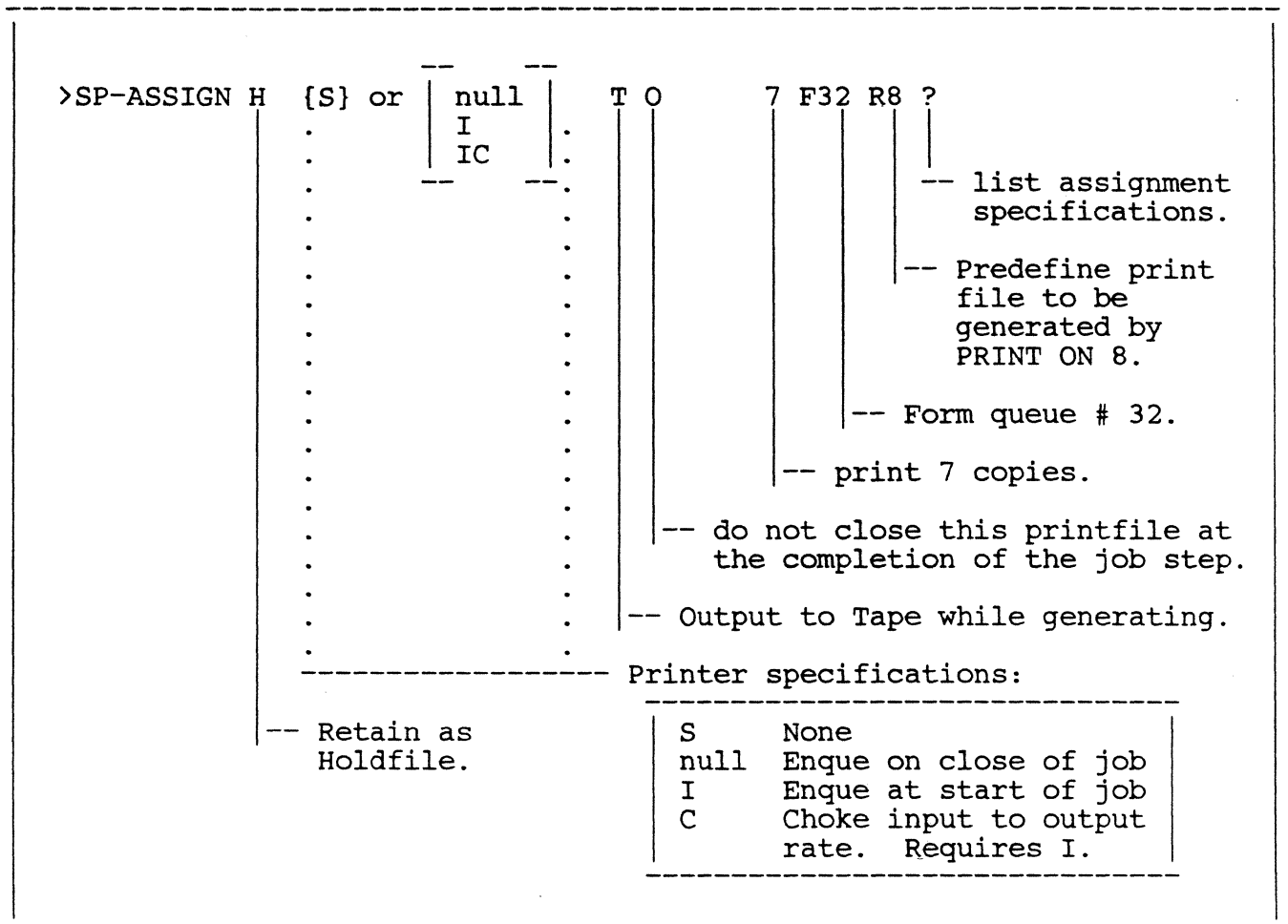
Predefinition	Rn	where n is the number of the printfile specified in the PRINT ON n statement in a basic program or from RUNOFF.

This option allows all of the other options. Clearly, however, if two or more printflles are sent to tape, their outputs will be commingled on the tape. Further, if more than one print file is to be choked, then there should be more than one printer available to process these jobs. Otherwise, the job will hang. Note that predefinition with the I option will cause the printer to attempt to output the printfile before the program commences to generate it. There is no harm in this, except that no other print jobs can be serviced by that printer until the program is complete. The execution of the SP-ASSIGN verb can be continued with R options to initialize several print files until the capacity of the input control record block is exhausted. There are about 60 input control block records available, depending on system configuration. Execution of the SP-ASSIGN verb without an R option will have the effect of closing all printfiles currently active.

7.3.3 The SP-OPEN and SP-CLOSE verbs.

These two verbs are superceeded by the SP-ASSIGN verb, but they are retained so that PROCs can declare a sequence of print jobs as one print file without having to know the other details of the SP-ASSIGNment currently necessary. The SP-OPEN simply turns on the bit associated with the "O" option of the SP-ASSIGN verb without disturbing any other parameters of the assignment. The SP-CLOSE causes the print file close sequence to take place without disturbing the assignment parameters, except to turn off the open bit.

7.3.4 THE GENERAL FORM OF THE SP-ASSIGN VERB.



The SP-ASSIGNMENT verb parameters.

Any or all of these parameters can be used together, except that the S parameter causes the I parameter to be without meaning, and that the absence of the I parameter or the presence of the H paramater causes the C parameter to be meaningless. The parameters can be in any order. The numbers associated with predefinition and form specification must be concatenated with their character keys, that is, R and F.

```
>SP-ASSIGN          -- the logon default setting:

LINE  STATUS COP  FORM
   #         IES    #

   5  P        1    0

   This will cause: Output to the printer with deletion
                        during printing;
                    The job to be enqueued upon completion;
                    In form Q 0;
                    One copy to be printed;
                    By a printer allocated to form 0, whenever
                        such a printer becomes available.
```

The SP-ASSIGN and the LOGON default.

```
>SP-ASSIGN HS3?

LINE  STATUS COP  FORM
   #         IES    #

   5  H        3    0

   This will cause: Retention of the printfile upon completion;
                    If the form and copy parameters are not
                        changed at SP-EDIT time, it will:
                    Print three copies;
                    On a printer allocated to form 0.
                    In order to change the assignment parameters
                        at SP-EDIT time, use the SP-ASSIGN verb,
                        followed by SP-EDIT with the R option.
```

The SP-ASSIGN H and S options.


```
>SP-ASSIGN TS?

LINE  STATUS COP  FORM
  #          IES    #

  5  T         1     0

  This will cause: The output generated by the process to
                       go directly to tape under the control
                       of the generating process.
                   There will be no spooler involvement.
```

The SP-ASSIGN T option.

```
>SP-ASSIGN CI?

LINE  STATUS COP  FORM
  #          IES    #

  5  PIC       1     0

  This will cause: The printfile to be linked onto output Q 0
                       as soon as the first line of output is
                       available.
                   If a printer is available for form Q 0, and idle,
                       it will commence to output the job.
                   In any case, should the generating process get 20
                       frames (a convenient constant) ahead of the
                       output processor, the generating process will
                       wait until the output processor has output
                       another frame.
                   Should the output processor catch up to the input
                       processor, it will wait until the output
                       processor has completed its current frame.
                   This option does not allow multiple copies.
                   This option is available for tape input to the
                       spooler.
```

The SP-ASSIGN C option.

```
>SP-ASSIGN HSR0

LINE   STATUS COP   FORM
   #          IES      #

   5  H         1      0
```

This causes the print file which will be generated by a PRINT or PRINT ON 0 statement in a succeeding basic program to be made into a hold file which, for the moment, will generate 1 copy on form 0.

```
>SP-ASSIGN TSR1

LINE   STATUS COP   FORM
   #          IES      #

   5  T         1      0
```

This causes the print file which will be generated by a PRINT ON 1 statement in a succeeding basic program to go directly to tape, without generating a print file on disc. It is necessary to have the tape drive available when the basic program is run. The copy and form parameters are irrelevant in this case.

```
>SP-ASSIGN CIR2F3

LINE   STATUS COP   FORM
   #          IES      #

   5  CI        1      3
```

This causes the print file which will be generated by a PRINT ON 2 statement in a succeeding basic program to go to the printer allocated to form 3, to be enqueued during the execution of this SP-ASSIGN verb, and to choke the output by the basic program to this print file to the output rate of the printer. The copy specification of 1 is forced by the C specification, and the I specification is required by the C specification.

An example of the use of SP-ASSIGN Rn.

7.4 HOLD FILE INTERROGATION: THE SP-EDIT VERB

The SP-EDIT verb is used to interrogate one or more hold files and and dispatch them in various directions.

The intent of the SP-EDIT verb is to allow access to print files which have been retained on disc for future output. It allows the user to obtain a print file and then direct it to the desired output device, or to delete it and release its storage space. The output devices available include any line printer or parallel printer on the system, the magnetic tape device, and data files. Print files may be specified by the entry number displayed at generation time, but there are other facilities available. Specification of destination is by means of parameters stored at print file generation time, and by the current SP-ASSIGNment of the line, but there are numerous options available to modify and override the default specifications. Execution of the process is by response to prompts, but there are options and facilities available to override these as well.

7.4.1 SP-EDIT OPTIONS.

The SP-EDIT verb previously allowed editing either everything, or one print file, n. It now allows greater selectivity. The previous form of the SP-EDIT verb presented print files generated by any and every account on the system for inspection. It now presents only those generated on the account onto which the SP-EDITing process is now logged. The system manager can override this convenience and security feature to some extent. The previous SP-EDIT verb required that the SP-EDITing process be SP-ASSIGNed to the desired output destination. The current form allows this as well as options specifying destination submitted to the SP-EDIT verb at SP-EDIT time.

7.4.1.1 PRINT FILE SELECTION OPTIONS

With no options in effect the SP-EDIT process will return all print files generated on the account onto which the SP-EDITing process is now logged. It is the same group of print files whose control records are displayed by the LISTPEQS verb with the A option.

This selection may be overridden by an account with SYS2 privileges, a condition which is normally limited to SYSPROG, the system manager. There are two approaches which may be taken. The U (Universal) option will return any hold file which is SP-EDITable, within the context of the other options which may be in effect. The other approach is to specify the account on which the print file was generated, which requires SYS2 privileges. It specifies that the SP-EDIT processor return only print files generated on the specified account, again within the context of other selection options which may be in effect.

Effectively, the U option obtains all available hold files, and as such is equivalent to the previous form of the SP-EDIT verb with no selection.

Hold file entries may also be referenced by entry number, n, or by a range of entry numbers, n-m. The numbers n and m must both be between 1 and 600

inclusive, and m must be greater than or equal to n, or an error message will ensue, and the SP-EDITing process will terminate. Selection of print files by the SP-EDITor in the presence of numeric options is in ascending order, starting with n and continuing through m. Each will be checked for admissability as an SP-EDITable hold file and then for admissability according to the generating account.

Selection may also be according to output queue specification. If the F (Form) option is selected, then n or n-m are taken to be output queue numbers rather than entry numbers. In this case n and m must be in the range of 0 through 125, with m greater than or equal to n, as above. In this case, selection will occur across all the available entries in the print file control block, using both output queue specification and generating account as selection criteria.

SP-EDIT AVAILABILITY

In order for a print file to be SP-EDITable, its control block record must pass certain system admissability tests. There must be a print file associated with the control block record, and it must be marked as a hold file. The SP-EDITor will then test for account name, entry number, or output queue number admissability. If the print file passes these tests, it will be checked to determine whether it is actually available, or unlocked. It is locked, not available, when it is being generated, when it is enqueued for output or is being output, or when it is being SP-EDITed. It is unlocked at the end of generation, at the end of output, or when it is removed from an output queue by the SP-KILL F verb, by the :STARTSPOOLER (C verb, or by a cold start. If an admissable hold file is discovered which is locked, the following message will appear: "ENTRY # n IS NOT AVAILABLE", and the SP-EDITor will search for the next entry. When an available entry is encountered, the print file is retreived, and the print file inspection and dispatch phase, which is the subject of most of this discussion of the SP-EDIT verb, is entered.

SP-EDIT TERMINATION MESSAGES

The SP-EDITing process will normally terminate with one of the two following messages. If entry number specifications were included and they do not exhaust the print file control block, then the message will be:

END OF REQUESTED PRINT FILES.

Otherwise, the message will be:

END OF PRINT FILE CONTROL BLOCK.

THE SP-EDIT LOOK, OR PRINT FILE PEEK.

There is a singular option which operates with the selection criteria which allows a peek at the first 500-odd bytes of any print file which is locked but not being output, or any print file marked as a hold file which is being output. This allows inspection of the print files in an output queue in order to identify them further than the LISTPEQS verb does. It is activated by specifying the L option. The L (look) option operates under the print file selection criteria discussed above, and enables the display of the first 500-odd bytes of print files which are locked. It specifically does not allow any manipulation of print files.

7.4.1.2 HOLD FILE DESTINATION OPTIONS.

The usual hold file manipulation will be discussed under the topic of the SP-EDIT prompt sequence below. Here we consider certain options submittable at verb activation time which modify the results of the normal SP-EDITing process.

The usual process at SPOOL time is to transfer the device specification from the SP-EDITing line's SP-ASSIGNment block, and either enque the print file to a printer queue, if the printer is specified, or to send the print file to tape under the control of the SP-EDITing process, if that is specified. In some situations under PROC control it is inconvenient to reSP-ASSIGN the line for the purpose of SP-EDITing, however, and it is occasionally convenient not to reassign while running interactively. For this purpose the P (Print) and T (Tape) options are available. The T option directs the output to tape, and the P option directs the output to the printer. They supersede the current SP-ASSIGNment specification, and the T option supersedes the P option.

In the case of the T option or SP-ASSIGNment T, the SP-EDITing process sends the print file to tape itself, and, as usual, the process checks tape attachment. If the tape is already attached, the process will proceed. If the tape is not attached, the process will attempt to attach it. If attachment is successful, the process will proceed. If the tape is attached to some other line, the attachment will be impossible, and, in the normal course of events processing will terminate with a tape not available message. In the case that the SP-EDITing is under PROC control it may be preferable to wait until the tape is available and then proceed. For this purpose the W option is supplied. It will cause the process to wait until the tape is available, spool the hold file to tape, and then return. The W option may be used any time that a hold file is being sent to tape. It has no effect otherwise.

TAPE RECORD SIZE WITH PRINT FILE TO TAPE

Specification of tape record size is available for all print file to tape operations, since all transfers to tape are done by the generating or SP-EDITing process. If there is no prior attachment by means of the T-ATT verb, the process will attach the tape on initiation. The tape block size will be the last tape block size used by the line, if tape has been used since logon time, or it will default to 500 bytes. It may be the case that there is a preferred size, in which case the use of the T-ATT verb with the desired parameter is recommended.

TAPE LABELS WITH PRINT FILE TO TAPE

In the normal course of events, a print file on tape will be preceeded with a tape label which includes the header "SPOOLER". Should it be desired that the print files on tape not be preceeded by a tape label, then the H (Header-suppress) option can be used. This may facilitate transfer of print files (or data) to foreign machines, or avoid having labels between each of several contiguous print files on a tape for use on an evolution machine. It is best that one not change tape record sizes between tape files without allowing a tape label at the beginning of the tape file with the new block size.

OUTPUT QUEUE AND COPY COUNT SPECIFICATION REPLACEMENT

The multiple printer spooler processor has several different output queues into which a print file may be enqueued, and it has the ability to output several copies of a report on a single activation. These are specified by the SP-ASSIGNment in effect when the print file control record is created. In the normal course of events, they are probably that which is desired. Therefore, they are the default output queue and copy count specifications. It may become convenient to change them, however. In this case, the SP-EDITor will obtain the new output queue and copy count specifications from the SP-ASSIGNment of the SP-EDITing process at SP-EDIT time. In order to cause this transfer of specifications, the R (replace) option is used.

GENERALIZED HOLD FILE MANIPULATION

The SP-EDIT process allows precise manipulation of print files. There occur times when less precision is necessary and more speed and less work are desired, as when it is desired to either spool or delete all the available hold files which can be selected under the selection techniques noted above. For this purpose, there is the M (Manage) option. It enables the S (Spool) and D (Delete) options. When the M option is in effect, and the S option is selected, then all selected print files will be spooled according to the destination options and specifications active at the time. When the M option is in effect and the D option is specified, then all print files passed by the selection criteria will be deleted and their storage space returned to overflow. Note that the use of the LISTPEQS verb is recommended, because the SP-EDITor will not pause to discuss the matter when the SP-EDITor is in this mode. The MD option is particularly recommended when an account name or an output queue specification defines the intended group of print files uniquely, although it can be used on print files identified by entry number using the n or the n-m options. All selection and destination options are available for use with the M _ option, but only the SPOOL Y or the DELETE Y alternatives of the SP-EDIT prompt sequence can be executed in this manner. The entry number of each affected print file will be sent to the terminal of the SP-EDITing process.

7.4.1.3 HOLD FILE TO DATA FILE OPTION

There is a further option associated with the hold file to data file capability which runs under the SPOOL prompt. In the normal case, the print file is transferred to a data file with one page per item, such that the trailing blank lines on each page are deleted from the data file. If the trailing blank lines are desired, there is a V (Vanilla) option which will cause all trailing blank lines on each page to be kept.

7.4.2 PROC CONTROL OF THE SP-EDIT PROCESS

The SP-EDIT process can be executed from PROC with stacked inputs in the same way that any other process which requests input is to be executed. Note that the PH comand will not avoid the prompts normally sent to the screen, which will generate at least some audit trail. The problem with

the use of SP-EDIT under PROC control is the matter of whether the PROC knows what it ought to SP-EDIT. For this purpose, each time a print file control record is created and its entry number is sent to the screen, the entry number is also placed in the PROC secondary input buffer. This allows a process running under PROC control to generate a hold file, and then immediately obtain the hold entry number or numbers from the secondary input buffer. The process may either pass an entry number to the SP-EDITor, or it may pass a collection of entry numbers to a BASIC program to file for future reference. The normal technique is to set the PROC input pointer to the secondary input buffer with an SS command, and then inspect and transfer the entries to the primary or secondary output buffer, as the case may be. Note that there may be error message numbers intermixed in the secondary output buffer, and that the SP-EDIT process does not allow a list of entry numbers. The intermixture is particularly likely if the print file generating process is the result of an instruction stream introduced from the secondary output buffer after a SELECT or an SSELECT introduced from the primary output buffer. The evanescence of secondary input buffer entries is one of the reasons for the P, T, and TW options.

If PROC control of the spooler structure is seriously contemplated, then there are two protocols which are recommended, and which depend on the selection options noted above. First, print files or print file classes may be identified by the generating account name. This must be a real system file account name rather than a CHARGE-TO name. It may be implimented either by requiring a LOGTO the generating account to spool the results of that account, or by logging the PROC to a SYS2 account to spool the results of named accounts in an orderly manner. Secondly, each print file class may be assigned a specific output queue number, so that each class can be selected and output together. Note that in each case the print files will be enqueued in ascending entry number sequence, which is not necessarily the sequence in which they were generated, or the sequence in which they are desired to be output.

7.4.3 THE SOURCE OF HOLD FILES

Hold files conventionally are the result of generating print files under an SP-ASSIGN H assignment specification. There are other conditions under which a print file directed to a printer may become a hold file. All existing print files which are directed to a printer which are not hold files either are already enqueued for output to a printer, or are in the process of generation and will be enqueued at the termination of generation under any completion state other than catestrophic failure of the system. Any of these may be disenqueued before it is being output by the use of the SP-KILL F verb. Upon disenqueuement, the print file will be returned in the form of a hold file available for later SP-EDITing. If the print file is being output, there are conditions under which it may be disenqueued and retained as a hold file. See the section on SP-KILL F. If a coldstart is executed, all salvageable non-hold files will be returned as hold files, and all output queues will be cleared. If the :STARTSPOOLER (C is executed, the same result will occur.

Selection options	Selection options
none	The default is to select all hold files generated on the account onto which you are currently logged.
U	Edit all hold files. Requires SYS2 privileges.
'accountname'	SP-EDIT print files generated on account 'accountname'. The single quotes are required. SYS2 privileges are required. The account name option will override the U option.
n	SP-EDIT print file number n if it is otherwise available.
n-m	SP-EDIT the print files whose entry numbers are n througn m inclusive, and which are otherwise avaliable. N and m must be within the range of 1 through 600 inclusive when referencing entry numbers, and m must be greater than or equal to n.
F	Used with n or n-m, causes the meaning of n and n-m to change to output queue numbers. Hold files whose output queue specication is n or is in the range n through m inclusive will be selected. Here, n and m must be in the range of 0 through 125 inclusive, and m must be greater than or equal to n.

SP-EDIT SELECTION OPTIONS

SP-EDIT	Will return available jobs generated by the account onto which you are now logged.
SP-EDIT U	Will return all available jobs.
SP-EDIT n	Will edit print file n, if it is yours.
SP-EDIT n-m	Will edit your print files between the numbers n and m inclusive, where n is greater than or equal to m.
SP-EDIT 'accountname'	Will return all jobs generated on account 'accountname', and requires SYS2 privileges.
SP-EDIT Fn	Will select all print files generated on this account and marked for output queue n.
SP-EDIT Fn-m	Will select all print files generated on this account marked for output queues n through m, where m is greater than or equal to n.

SP-EDIT selection -- simple examples.

Print file inspection options.

L	Inspects an enqueued print file. This option accepts all selection options and ignores all manipulation options.
O	In conjunction with the L option, the O option allows peeking at hold files which are being output.

Hold file manipulation options.

R	Uses the current SP-ASSIGNment parameters for the form specification and copy count.
P	Forces the print file to the printer. Superseeds the line's current SP-ASSIGNment output specification.
T	Forces the print file to tape, as above. Supersedes a P option.

Hold file to tape sub-options.

H	Causes no tape label to be put on a tape when the print file destination is tape. The option is effective with either the SP-ASSIGN tape specification or the SP-EDIT tape specification.
W	Causes the SP-EDITing process to wait for the tape drive to be available. Used in conjunction with the T option to simulate the prior spooler's system for tape print file generation.

Force option

M	Manager -- allow multiple hold file manipulations without intervention at each prompt, according to the two following options:
S	Spool each hold file selected.
D	Delete each hold file selected.

Hold file to terminal option

N	Causes ouptput to the terminal to run continuously across page breaks. The option with the "T" response is equivalent to the "TN" response. It has no effect with respect to any other destination.

Hold file to data file option

V	Vanilla hold file to data file conversion: do not remove trailing blank lines from each page during conversion.

SP-EDIT	The default is to output the hold file to tape if the SP-EDITing line's current SP-ASSIGNment so specifies, or to enqueue the hold file for output to a printer. The hold file is enqueued in the output queue specified by the entry in the print file control block record, not in the SP-EDITing line's current SP-ASSIGNment. The number of copies printed will be the number specified in the control record.
SP-EDIT L	Will allow you to peek at jobs which are enqueued for output but are not being output. This option supersedes all other destination specifications and options.
SP-EDIT LO	Will allow you to look at the first 500-odd bytes of print files which are being output and are marked as hold files.
Options related to printer output	
SP-EDIT R	Will function as SP-EDIT, and transfer the copy count and form number in your current assignment specification to the print file control record and enque the print file accordingly. This option has no effect if the destination of the print file is tape.
SP-EDIT P	Force the print file to be enqueued for output to the printer.
Options related to tape output	
SP-EDIT T	Force the output to the tape if the tape drive is available. Otherwise, the process will terminate with an error message.
SP-EDIT W	As above, if the SP-EDITing line's output assignment is to tape. Otherwise, the W option has no effect.
SP-EDIT H	If the SP-EDITing line's current output assignment is to tape, then the standard tape label will not be written. The default is to write the label. If the destination of the print file is not to tape, then the option is irrelevant.
SP-EDIT TW	As above, except that the process will continue to attempt to attach the tape rather than terminating. It will continue to attempt attachment indefinitely.
SP-EDIT TH	Forces output to tape, does not write a label.
SP-EDIT THW	Forces the print file to tape, waits for the tape drive to be available, does not write a tape label.

SP-EDIT hold file manipulation options -- simple examples part 1.

Forcing options

SP-EDIT MS — Force all selected print file to be spooled to the device specified by the SP-EDITing line's current SP-ASSIGnment specification, or by the options in effect.

SP-EDIT MD — Force all selected print files to be deleted without passing go or collecting the traditional, if shrinking, 200 dollars. The process is breakable only between deletions.

Hold file to terminal option

SP-EDIT N — Causes the T response to the T option to behave like the TN response. The N option has no other effect.

Hold file to data item option.

SP-EDIT V — In conjunction with an F response to the SPOOL prompt discussed below, this option will override the default deletion of trailing blank lines at the end of each page.

SP-EDIT hold file manipulation options -- simple examples part 2.

SP-EDIT RU3 — Will SP-EDIT entry 3, generated by any account and utilize the SP-EDITing process's form and copy count specifications.

SP-EDIT 7-11 — Will send each print file generated by the account onto which the process is now logged with entry numbers 7 through 11 inclusive to tape if the current assignment is HS.

SP-EDIT 10-20 — Will obtain all allowable hold files with entry numbers 10 through 20 inclusive in ascending order.

SP-EDIT R 'accountname' — Will obtain all available hold files generated on account 'accountname' and transfer the SP-EDITing process's copy count and form specification to any of the hold files which are spooled to the printer.

General examples of the SP-EDIT verb.

SP-EDIT 4LU	Will look at the first 500-odd bytes of print file number 4 created on any account if the user has SYS2 privileges and it is not being output.
SP-EDIT 4LOU	Will do as above, and include the case that print file 4 is a hold file which is being output. If it is not a hold file, it will not be displayed.
SP-EDIT F3MS	Will spool all print files created on this account and specified to be enqueued on output queue 3, according to the SP-EDITing line's current output specification.
SP-EDIT F4 P MS 'ACCOUNTING'	Will enqueue all available print files created by the account ACCOUNTING, specified to be enqueued in output queue 4, to the printer irrespective of the SP-EDITing line's current SP-ASSIGNment specification, without recourse to the interactive prompts, in the natural order of the print file control records, if the SP-EDITing account has SYS2 privileges.
SP-EDIT U F 10-20 MD	Will delete all available print files, created on any account and specified to be enqueued onto output queues 10 through 20 inclusive, in the natural order, without recourse to the prompts, if the SP-EDITing account has SYS2 privileges.
SP-EDIT MD	Will delete all available print files created by this account.

General examples of the SP-EDIT verb.

7.4.4 The SP-EDIT prompt sequence.

The SP-EDIT verb is interactive. This section discusses the prompts given and responses allowed.

```
ENTRY # nnn
DISPLAY? (Y/N/S/D/X/(CR))

          Y       Display.
          N       Skip to STRING.
          S       Skip to SPOOL.
          D       Skip to DELETE.
          X       Terminate SP-EDIT.
          (CR)    Skip to next print file.
                  Any other response will skip to STRING.

STRING-

          (cr)    Skip to SPOOL.
          text    Scan print file to 'text'.

SPOOL (Y/N=CR/T/TN/F)?

          Y       Enque for output to a printer
                  or output to tape directly.
          N       Skip to DELETE.
          (cr )   Skip to DELETE.
          T       Output to user's terminal.
                  Roadblock at the end of each page.
          TN      Output to user's terminal without
                  roadblock.
          F       Convert to data file item set.

DELETE (Y/N=CR)?

          Y       Release remaining storage to overflow
          N       Skip to next print file.
          (cr)    Skip to next print file.
                  Any other response will skip to
                  the next print file.
```

The SP-EDIT prompts.

Note that the ALIGN prompt has disappeared. See the STARTPTR verb.

7.4.5 THE DISPLAY PROMPT.

The SP-EDITor will specify: ENTRY # nnn, and will then output:

DISPLAY? (Y/N/S/D/X/(cr))

The suggested responses have the following meanings:

Y says yes to the DISPLAY question, and the processor will proceed to display as many lines as are required to output the first 500 bytes of the report. It will go beyond the 500th byte in order to complete the last line of the display. It will display the whole report if the report is less than 500 bytes.

If the L option is in effect, the DISPLAY prompt will be presented. The carriage return response will cause selection of the next hold file, the X response will cause the process to exit to TCL, and any other response will generate the 500-odd byte display of any print file which is not being output. If the print file is being output, and the O option is in effect, and the print file is marked as a hold file, then the first 500-odd bytes will be displayed. Otherwise the message:

BEING OUTPUT

will be displayed, and the process will continue to the next selected hold file or to TCL. If the requested fragment of the print file is displayed, the prompt NEXT? will occur. When it is desired to proceed to the next hold file or TCL, then a carriage return will cause the process to recommence.

N will cause the processor to skip the display routine and proceed directly to the STRING- prompt.

S will skip both the display and string routines, and proceed directly to the SPOOL? prompt.

D will skip all of the above and proceed directly to the DELETE prompt

X will leave the SP-EDITing process immediately.

(CR) will skip to the next requested hold file, if any.

7.4.6 THE STRING PROMPT.

STRING-

The intent of the function available at the STRING- prompt is to allow the continuation of the printing of a report which was previously interrupted, as by a paper jam, without having to reprint that which was already printed. A carriage return will skip the function. Any other input is construed as a litteral string from the prompt to the carriage return, including blanks. The processor will then scan the hold file, starting from the top, until it encounters the first instance of the desired string. It will then set the beginning-of-report address to the beginning of the line in which the string was found, and deliver the result to the SPOOL prompt. If the report in question was paginated by the standard

system output processor (any case other than line-counting in BASIC, an assembler routine written by a user, or a print file generated with a page length of zero as specified by TERM), then it is sufficient to align the paper to the standard top-of-form for the printer and initiate output from the SPOOL prompt in order to obtain proper alignment. This will work if the string sought is on a page prior to the desired output. When the printer encounters the top-of-form byte at the top of the next page, the printer will eject paper to the top of the page, yeilding alignment of the paper and the text. If the job was generated without top-of-form characters, the paper will have to be aligned to the correct relative location on the paper. See the STARTPTR verb for the alignment proceedure.

7.4.7 THE SPOOL PROMPT.

The SPOOL prompt has the form:

SPOOL (Y/N=CR/T/TN/F)?

The responses to the SPOOL prompt, in conjunction with the process's current assignment and the options on the SP-EDIT verb, cause the print file to go to the specified destination.

7.4.7.1 THE Y RESPONSE.

Y enqueues the job for output by a printer if the SP-ASSIGNment of the SP-EDITing process so specifies or the P option is in effect, or sends the print file to tape under the control of the SP-EDITing process if the SP-ASSIGNment of the SP-EDITing process so specifies, or the T option is in effect. Note that options supersede SP-ASSIGNment specifications, and the T option or specification supersedes the P option or specification. Also note that the SPOOL Y may be forced by the use of the MS option. If neither the P nor the T option nor an SP-ASSIGNment specification to printer or tape is in effect, then the following message will be output to the terminal and the process will return to TCL:

YOUR OUPUT SPECIFICATION IS NO OUTPUT.
REASSIGN YOUR LINE IF YOU WISH TO OUTPUT A HOLDFILE.

When a job is enqueued, the SP-EDITing process executes the enquement, alerts the spooler, and returns to the next desired hold file, if any. The print file is enqueued in the output queue specified in the print file control record as specified in the SP-ASSIGNment form number in effect at print file generation time. RThis may be overridden by the option, which will transfer the SP-EDITing line's current form number specification to the print file control record, and enqueue the print file in that output queue. The changed form number will persist. The next time that the print file is SP-EDITed with out an R option in effect, the print file will be enqueued in the output queue specified by the new form number specified by the last SP-EDIT operation with the R option in effect.

If the process is to copy the hold file to tape, the following sequence will occur. The process will attempt to attach the tape. If this is not possible, the message

TAPE ATTACHED TO LINE nn

will be output to the screen and the process will return to TCL. If the W option is in effect, the process will wait until the tape becomes available. If the tape is not attached already, and it is available, the following message will be output:

TAPE ATTACHED BLOCK LENGTH IS nnnn.

The tape block length will be the system default 4000 byte records. If the tape was already attached by means of the T-ATT verb, the same message will be output, and the block length will be that set at T-ATT time. See the discussion of tape record size for further considerations thereon.

The SP-EDITing process will then put a tape label on the tape which will include the text field "SPOOLER". If the tape is to have no label, then use the H option with the SP-EDIT verb. This will cause suppression of the tape label. Note that this is the HDR-SUPP option in ENGLISH, and that the use of the H option with any verb which generates a print file in conjunction with a T assignment specified will cause tape label suppression. If the verb is an ENGLISH verb, the HDR-SUPP connective will achieve the same result. The SP-EDITing process will then proceed to copy the hold file to tape, and will return to the DELETE prompt upon completion of the tape output.

7.4.7.2 THE T RESPONSE.

T will cause the contents of the hold file to be printed on the terminal of the SP-EDITing process, and will request a carriage return at the end of each page in order to continue to the next. In place of the carriage return the characters U, T, or X may be used. Use of the character U will cause the current page to be repeated; use of the U at the end of this page will cause the previous page to be displayed. Its predecessor is not immediately available. Use of the T will cause the report to start again from the top. Use of the X will cause termination of the spooling to the terminal and a return to the SP-EDITor process to the SPOOL prompt. Note that a carriage return may be necessary to obtain the first page, and that occasionally page p-2 may appear instead of p-1 on the second U command. The SP-EDITing will return to the SPOOL prompt.

7.4.7.3 THE TN RESPONSE.

TN will cause the contents of the hold file to be output to the terminal of the SP-EDITing process, but will not pause at the end of each page, consequently, the above control characters are unavailable in the nopage case.

It is possible to emulate a serial printer using a normally-logged-on terminal using the SPOOL TN response. The processor will count lines and execute line-feed to the bottom of the current page when it encounters a form-feed character. Successful execution of this technique requires the execution of the TERM verb in order to specify the actual page length of the paper onto which the print file is to be spooled. In other words, if you have 11 inch long paper, and the printing terminal is set for 6 lines

per inch, the TERM verb must be used to tell your process that there are 66 lines per page, as follows:

TERM ,66

The SPOOL TN process will not send form-feed characters to a printing terminal in the way that the serial printer process would prefer to do.

Be sure to reset the TERM before you commence to do normal processing.

The SP-EDITing process will return to the SPOOL prompt.

7.4.7.4 THE F RESPONSE.

F will allow the transfer of a hold file to an item or a series of items in a file in RUNOFF format. This can only be accomplished when the SP-EDITing process is logged onto the account which created the hold file. The process will prompt with FILE NAME-, at which time the name of the file into which the item or items are to be inserted is input, and then with INITIAL ITEM-, at which time the name of the first item of the string of items to be generated from the hold file is to be input. An incorrect file name will cause termination with the usual error message. The item name may be an existing item, at which time the existing item will be over-written.

The hold file to file translation is accomplished by putting a leading .bp .nf into the head of the item, and then by copying the contents of the hold file into the item, with two modifications. First, carriage return, line feed sequences are removed and an attribute mark is inserted in their place. Second, upon a page break, the processor will terminate the current item with a .CHAIN ITEM-NAMEnnnn, file the current item, and initialize the next item. The sequence "nnnn" concatenated to ITEM-NAME is a member of the sequence 0001, 0002, 0003, ... The first item in the string has the name ITEM-NAME and chains to ITEM-NAME0001; the second item in the string has the name ITEM-NAME0001 and chains to ITEM-NAME0002, and so on. The terminal item has no chain statement in it. Clearly, this set of item ids will sort into sequence. Hold files which do not have top-of-form characters in them or which have very long pages will be blocked into items about 12000 bytes long.

The use of this facility includes the ability to merge anything printable into documentation, to merge ENGLISH reports into RUNOFF reports within the body of the RUNOFF text, and to retain any output file as part of the saved files in the system.

The proceedure does not modify the print file itself except to delete any trailing blank lines in the text. These trailing blank lines may be retained if the V option is selected when initiating the SP-EDIT verb. Upon completion the process returns to the DELETE prompt in the SP-EDIT process.

It is necessary to be logged onto the account which created the print file in order to execute this process.

7.4.8 THE DELETE PROMPT.

DELETE (Y/N=CR)?

The only action which will cause deletion of the print file is the execution of the "Y" response. Any other response will obtain the next hold file desired, if any. Deletion is executed by the SP-EDITing process. Once it is initiated, it is not breakable, in order to protect the overflow table. Upon completion of the release of storage to the system and the reinitiation of the print file control block, the process will proceed to the next desired hold file, if any. Note that the reinitialized print file control block will be unchanged under listing by the LISTPEQS verb, except that it is marked available, until such time as it is reused. This gives some chance of discovering that the print file, which apparantly disappeared, did so because it was deleted.

Print files may be systematically deleted by the use of the MD option on the SP-EDIT. The prompts will not be seen, but the print files will be deleted. The process is not breakable during the deletion of a print file. It will respond to the break key between print files.

7.5 THE PRINTER CONTROL VERBS. THE PRINTER CONTROL VERBS.

The following sections discuss the verbs which allocate, start and stop printers, and which terminate print jobs and printers.

Three verbs are supplied to start, stop, and delete printers from the spooler system. The verb which starts printers, the STARTPTR verb, defines the printer. It specifies the printer ordinal, the device, the output queues to consume, and the inter-job page skip to execute. It also executes the form-alignment process when desired. The STOPPTR verb simply flags the stopped printer to terminate output at the end of its current print file. The D option used with the SP-KILL verb causes deletion of the specified printer from the spooler system. Associated with these verbs are the other uses of the SP-KILL verb, which control print files enqueued for output.

7.6 THE STARTPTR VERB. THE STARTPTR VERB.

The STARTPTR verb specifies a printer and allocates it to specific output queues, and starts it.

The STARTPTR verb is the primary control processor of this spooler. It can be executed either upon initiation of the spooler, or after a STOPPTR has been executed, and the printer has stopped.

The STARTPTR verb specifies 1) the printer number of a device, 2) the output queue or output queues upon which it is working, 3) the page skip to execute at the end of each print file, 4) the type of printer, P for parallel or S for serial, 5) the device number for parallel printers or the line # for serial printers, and 6) whether this is an alignment, which is no longer in the SP-EDIT process. Further, if the target printer is a serial printer which does not recognize a X'0C' as a page-eject command, then the X option is available to specify this, and the number of lines per page may be specified as a numeric option.

Note that in the case of a parallel printer, the printer number and the printer device number should be the same. The system will take the printer number as the device number in any case. It requires that there be a device number, but it will then ignore it. Also note that the system can and may be defined for more than four parallel printers. If your system has the hardware to support two printers and the software to allow four printers, and you start a third parallel printer, printer 2 on device 2, the worst that will happen is that a 'no printer controller' message will occur for that printer under SP-STATUS, and that the printer will grab a non-hold file and refuse to let go of it. An SP-KILL D2 and a :STARTSPOOLER C should attend to the situation. Then SP-EDIT the print file, which should now be a hold file, to respool or delete it, and reinitialize the other printers as desired. Also, reSP-ASSIGN all the lines on the system.

```
STARTPTR n,f,p,Tm,A (O)              for a single output queue, or

STARTPTR n,(fl,f2[,f3]),p,Tm,A (O)   for multiple output queues,

where

  n is the printer ordinal specified for the printer
    where n is between 0 and 19, inclusive.

  f or fi is the output queue Q number, where
    where f or fi is between 0 and 125, inclusive.

  p is the number of pages to skip, where p is between 0 and 9,
    inclusive.

  T is the printer type, P for parallel or S for serial printers.
    Note that a parallel printer runs off a GPIO board, and
    that a serial printer runs off an eight-way board.

  m is the line number for serial printers or the physical
    device ordinal for parallel printers, where m is 0 to 3 for
    parallel printers, and is one of the legal port numbers for
    serial printers.

  A initiates the alignment process.

  O options, which may be:

    X indicates that this serial printer does not recognize a
      X'0C' as a page-eject command, and that the printing
      process must count lines within the page, and emit the
      correct number of blank lines when each page-eject
      command occurs.

    N where 'N' is numeric, indicates the number of lines per
      page.  It is only effective with the X option.  If the
      X option is specified and there is no numeric
      specification, then the page length will default to 66
      lines.

    S option indicates that the initial form-feed command at
      the initiation of a print file is to be ignored by a
      serial printer.  It is not available for parallel
      printers.  It is best used in the case that the inter-job
      page-eject count is nonzero.  It is intended to be used
      in a packing-slip, picking-slip environment.
```

The form of the STARTPTR verb.

The syntax of the STARTPTR verb requires that the printer number be specified on each use of the verb, hence the example of the minimal activation. At initialization, after a STARTSPOOLER (I) or a cold start, the page skip, type of printer, and device address or line specifier, must be included in the use of the verb. Once printer type, address, and page skip are specified, then they will persist until changed.

Note that the device ordinal controls the printer number in the case of parallel printers. The printer allocated P0 will consume the forms specified in the control block for printer 0 without respect to the printer ordinal specified.

Use of the ALIGN option, A, will produce the prompt LINES?, which is asking for the number of lines to output on each alignment attempt. After each trial, the prompt AGAIN(Y/T/N) will be forthcoming. A response of Y will cause another alignment attempt. A response of T will cause the alignment attempt to be terminated, and will leave the printer in question stopped. Any modified printer parameters will be stored in the printer control block, however. Any response other than Y or T will cause the real print file to be printed. During the course of the alignment, the printer is not attached to the line which is doing the alignment, but the printer will not do anything else, either.

7.6.1 EXAMPLES OF THE STARTPTR VERB.

```
>STARTPTR 0,0,1,P0
          | | | |-- specifies parallel printer on device address
          | | |     05.  This printer will consume the queues
          | | |     specified for printer 0.
          | | |
          | | |-- eject one page after each job.
          | |
          | |-- specifies that the printer will print jobs in
          |     output queue 0.
          |
          |--this is printer 0 in the LISTPTR display.
```

Form of the STARTPTR verb to be used for a default system.

```
>STARTPTR 1,(0,3,11),2,P1,A
          |  |       | | | |-- specifies alignment at start.
          |  |       | | |
          |  |       | | |-- specifies parallel printer, using the
          |  |       | |     second system device address, 06.
          |  |       | |
          |  |       | |-- eject two pages after each job.
          |  |       |
          |  |-------- this printer is to print output queues 0, 3
          |           and 11.
          |
          |-- this is printer number 1 in the LISTPTR display.
```

General form of the STARTPTR verb.

```
>STARTPTR 1
         |-- specifies printer 1.  This is a restart without
             parameter changes after a stop.
```

Minimal form of the STARTPTR verb.

```
>STARTPTR 3,(2,4
          | |-- specifies forms 2 and 4.
          |-- specifies printer 3.
```

Minimal form of the STARTPTR verb with two output queues to be printed.

```
STARTPTR 1,2,,,A
         | |   |-- Alignment
         | |-- of job on output queue 2
         |-- on printer 1.
```

Printer restart with alignment.

```
>STARTPTR 7,12,2,S23 (X 51
          | |  | |    | | |-- Page length is 51 lines.
          | |  | |    |-- Specifies line counting since this
          | |  | |       serial printer does not recognize the
          | |  | |       X'0C' character as a form-feed.  Used
          | |  | |        with the '51' numeric option.
          | |  | |-- specifies serial printer on line 23.
          | |  |-- eject two pages after each job.
          | |-- specifies that the printer will print jobs in
          |     output queue 12.
          |--this is printer 7 in the LISTPTR display.
```

Form of the STARTPTR verb to be used for serial printers which do not recognize X'0C' as a form-feed character.

7.6.2 THE PRINT FILE SCHEDULING ALGORITHM.

It is possible to allocate several printers to a single output queue as well as to assign up to three output queues to a single printer. In either case the order of execution of jobs within an output queue and amongst output queues becomes a matter of interest.

When a printer locates a print file available for output, the printer process marks the print file as being output, but leaves it enqueued for output until the task is complete. Print files are left enqueued for output so that, if there is a system difficulty which requires the use of the :STARTSPOOLER verb, the spooler system can restart from a point prior to whatever problem occured. A discussion of the disposition of print files under :STARTSPOOLER and COLDSTART conditions will be found in the section on the :STARTSPOOLER verb.

When a printer completes a print file task, it will remove the print file control record from the output queue. Completion is defined as either normal completion, or termination under STOPPTR conditions, or termination under SP-KILL conditions.

The print file task initiation algorithm searches the output queue specified first in the STARTPTR verb until it encounters the first job not being output. If there are no print files available to be printed in the first-named output queue, the printer will search the second-named output queue in order, and then the third. It will not persist in the output queue whence the last job came. It is therefore possible to set priorities on print jobs using a single printer. The highest-priority jobs are sent to the first-named output queue, the second-priority jobs to the second-named output queue, and the lowest-priority to the third-named output queue. The priority structure can be reset by stopping and then restarting the printer. Jobs sent to other output queues will not appear until the printer is allocated to those output queues. It is the case that only one printer can look in the output queues at a given time.

7.6.3 STARTPTR ERROR MESSAGES

The STARTPTR process has a full set of error messages. In the order in which they are encountered, they are:

NULL PRINTER NUMBER`

This will occur when the first parameter, the printer number, is either null or non-numeric.

PRINTER NUMBER TOO BIG

This will occur when the printer number is larger than 19.

NO FORM NUMBER

This will occur when the second parameter, form number, is null or non-numeric. It is required for initialization.

ILLEGAL CHARACTER

This will occur when the parser encounters an unexpected character in one of several different circumstances.

PRINTER MUST BE STOPPED

This indicates that the printer which is being started either has not been stopped by the use of the STOPPTR verb, or has been set to stop but is still outputting a job. If it is desired to stop the printer immediately, use the SP-KILL verb. If it appears that the printer is unstoppable, then it may become necessary to use the SP-KILL verb with the D option.

FORM NUMBER TOO BIG -- EXCEEDS 125

This may apply to any of the output queue specification parameters.

TOO MANY PAGES IN THE PAGE SKIP -- EXCEEDS 9

This should be self-explanatory.

NEGATIVE NUMBER

As above, but probably indicates an aberation of some sort.

TOO MANY PRINT FILES

A maximum of three forms may be specified. Note again, that if there is more than one form specified, the group of forms must be placed in parantheses, that the right parentheses may not be omitted, and that the paranetheses are to be placed within commas. The exceptional case occurs upon restart, in which case the trailing parenthesis and comma may be left off if the page skip parameter is not to be changed.

ILLEGAL PRINTER TYPE -- NOT P OR S

The printer types may be only P for parallel or S for serial.

ILLEGAL LINE NUMBER OR PARALLEL PRINTER NUMBER

Serial printers may be allocated only to legal ports on the system. The spooler process may not be used as a serial printer. Parallel printers must be specified as 0, 1, 2, or 3. Note that you can start a parallel printer as parallel printer 1 even though there is not a GPIO board in the computer with device address 6. This will cause the spooler to waste a lot of time looking for it. The search can be terminated with an STOPPTR and three SP-KILLs, or with an SP-KILL with the D option. The latter approach may leave a print file in an ambiguous state.

ILLEGAL PARALLEL PRINTER NUMBER

Must be 0, 1, 2, or 3.

ILLEGAL SERIAL PRINTER NUMBER

Must be between 0 and 19 inclusive.

ALLOCATION ATTEMPTED ON UNINITIALIZED PRINTER

The printer has not been fully started yet. Tell it what its page skip is, what type it is, and what its physical device specification is.

YOU ARE ATTEMPTING TO START PRINTER nn ON LINE mm, WHICH IS NOT STOPPED

The printer control block, nn, is allowable in this case, but the line, mm, is still active, indicating that the stop/active flags are not set the same way in the two control blocks. The SP-KILL verb with the D option is probably called for in this case.

YOUR ALIGN PROCESS WAS JUST ABORTED BY SOMEONE.
START THE ALIGN PROCESS OVER.

An SP-KILL was apparantly executed on the printer which is the object of the align.

THERE IS NO JOB ENQUEUED FOR OUTPUT ON THE FORMS YOU SPECIFIED.
THEREFORE, ALIGNMENT IS IMPOSSIBLE.

In this case, execute an SP-STATUS or a LISTPTR n, where n is the logical address of the printer specified as the first parameter in the STARTPTR verb, to see what forms were specified. Then used the LISTPEQS verb with the F option to see what is enqueued in which queues and what their statuses are. If all appears correct, disenqueue them with the SP-KILL verb with the F option, and reenque them with the SP-EDIT processor.

THE PRINTER CONTROL BLOCK HAS BEEN INITIALIZED.

This indicates success for the STARTPTR process.

THE DEVICE OR LINE WHICH YOU SPECIFIED IS BEING USED
AS ANOTHER PRINTER ON THE SYSTEM.

Another printer control block has control of the device or line.

7.7 THE STOPPTR VERB

The STOPPTR flags the specified printer that it is to stop at the end of the current print file.

The STOPPTR verb is analogous to the old SP-STOP verb, execpt that it is intended to stop a printer, rather than the spooler. It requires a numerical argument in the same way that the SP-KILL verb does. The STOPPTR verb sets a flag which causes the printer to stop after it has completed outputting its current print file. The process can be expedited by executing a SP-KILL to that printer to complete the print file sooner. If the current print file is specified to print multiple copies of the report, then the printer will stop at the completion of the current copy.

If the STOPPTR verb is being used under PROC control with the STARTPTR verb, the STARTPTR verb will terminate unsuccessfully if the printer is still active when the start is attempted. Two facilities are available to control the timing of the execution of the STARTPTR verb. First, there is the W option, which causes the STOPPTR verb to wait until the printer has become inactive before it returns to TCL and thence to the initiating PROC. If the W option is used, the printer will be stopped and inactive when control returns to the user. Second, the messages which the verb sends are from the ERRMSG file and have error numbers which are conveniently stored in the PROC secondary input buffer. Error message number 1171 says that the printer is inactive and error message number 1172 says that the printer is still active. The E form in PROC cannot in general be used in this case, because it references only the first element of the PROC secondary input buffer.

To access the PROC secondary input buffer, execute an SS to set the PROC input pointer to this buffer, and then execute a scan of the buffer. The intent of the scan is to test for the ERRMSG number indicating the state of the printer process of interest to the PROC. For a complete discussion, see the section on handling error messages. Using the error message numbers rather than the W option allows the PROC to continue with other processing, and then to return to start the printer at a later time. Bear in mind that the PROC secondary input buffer is evanescent. It will disappear at the next verb execution.

Note that SYS2 privileges are required for the STOPPTR verb.

>STOPPTR n{-m}	Where n and m are a legal printer numbers between 0 and 19 inclusive.
>STOPPTR B	Will stop all printers.
>STOPPTR W	Will cause the process to wait until the printer has completed any print file and is inactive.

The general form of the STOPPTR verb.

>STOPPTR 1	Will set printer 1 to stop at the completion of its current job
>STOPPTR 3-5	Will set printers 3 through 5 inclusive to stop on completion of their current jobs.
>STOPPTR B	Will set all printers to stop as above.
>STOPPTR 1W	Will set printer 1 to stop and will wait until printer 1 becomes inactive before the process retuns to TCL.

Examples of the STOPPTR verb.

>STOPPTR	Is erroneous because no printer number has been specified.

Erroneous use of the STOPPTR verb.

1171	The printer is inactive
1172	The printer is active.
1174	The printer is unallocated.

STOPPTR error message numbers.

7.7.1 STOPPTR ERROR MESSAGES.

The following error messages will occur upon successful completion:

PRINTER # nn SET TO STOP AND IS INACTIVE.

In this case, the STARTPTR verb may be executed on this printer.

PRINTER # nn SET TO STOP BUT IS STILL ACTIVE.

In this case the stop flag has been set but the printer is still outputting a job. The printer will go to inactive status upon completion of the job, a condition which will be specified by the SP-STATUS and LISTPTR verbs in their normal course, as well as the STARTPTR verb as an error message.

The following messages indicate errors:

ILLEGAL PRINTER NUMBER. MUST BE BETWEEN 0 AND 7 INCLUSIVE.

This will occur if n lies outside of the legal range, or if n is not used. There is no default to zero.

PRINTER # nn CONTROL BLOCK HAMMERED. CLEARED TO NULL.

This message will occur if the contents of the printer control block does not pass certain validation requirements. The message, and variants thereon, can occur from other verbs which utilize the printer control block, because it is checked for validity upon each use.

The effect is to deallocate the printer control block, but to ignore the printer entirely. In this case, the process or subprocess which is acting as the printer may continue to sleep, print or commit mayhem. It cannot be stopped in an orderly manner by means of the STOPPTR and SP-KILL verbs, or even in a disorderly manner with the SP-KILL verb with the D option. It can be reinitialized with the initialization form of the STARTPTR verb, however; or the :STARTSPOOLER verb can be used to clear all printer processes and clean the output queues.

An attempt to execute the STOPPTR verb when logged onto an account with a privilege level lower than SYS2 will yeild the following message, and a return to TCL.

YOUR SYSTEM PRIVILEGE LEVEL IS NOT SUFFICIENT FOR THIS STATEMENT.

7.8 The SP-KILL verb and its extensions.

The SP-KILL verb serves to terminate various elements of the spooler structure. It is used to terminate print jobs immediately, to disenqueue print files, and to remove printers from the spooler system. Control is by options.

The SP-KILL verb has acquired several functions beyond simply terminating the output of the current job on the printer, and several options to control the verb. The SP-KILL can now disenqueue print files enqueued for output, and remove printers from the system entirely.

The SP-KILL verb is unchanged when used for the termination of the output or the current print file, excepting that the printer executing the job must be specified. It will be 0 or 1 for parallel printers and between 2 and 7 inclusive when the serial printer processors are working. Use the LISTPTR and LISTPEQS verbs to find out what is happening.

Note that the option strings may be put inside of parentheses as well as be used without parantheses. See the section on options.

Which elements:	
n	Specifies element n, where n is between 0 and 7, inclusive, for printers, or is between 1 and the last active print file on the system, which must be less than or equal to 600, for print files.
n-m	Specifies elements from n through m inclusive, where n is less than or equal to m and m is in the same range as n, above.
B	Specifies all elements possible.
What to kill:	
null	Specifies termination of current print job on specified printer or printers.
F	Specifies unlinking of print files.
D	Specifies deletion of a printer from the system.
Suboptions:	
N	Specifies no abort message when killing the output on a printer.
O	Specifies unlinking of a print file which is being output, where possible.
A	Specifies unlinking of print files created on the account onto which you are now logged, or the termination of output of print files of the same sort.

SP-KILL options.

7.8.1 PRINT FILE TERMINATION.

This is the classical use of the SP-KILL verb. It is used to terminate the print file being printed on a given printer. The verb requires an option specifying the logical number of the printer to which that job is going, or the range of printers to kill, or the B option to specify the termination of print files going to all printers. Generally, the user can kill only print files generated on the account onto which he is currently logged. SYS2 privileges for the account allow termination of any print file.

After the execution of the SP-KILL verb, the printer will terminate the print file with the message ABORT!, release the print file if release was specified, or make available for SP-EDITing. The printer will then proceed to the next print file in its assigned queues which is available

for output, and commence to output the print file, unless the STOPPTR verb has been executed for this printer, in which case the printer will stop and await reassignment.

The suboption N suppresses the message ABORT! which is normally placed one line after the point where the text of the report was terminated.

Note that executing SP-KILL n on print file n will cause its control record to be marked as aborted, which will appear on the status displayed by the LISPTEQS verb for print file n as an X.

The normal form of print file termination is:

SP-KILL 3	This will cause termination of the print file going to printer # 3 if either the account doing the SP-KILL has SYS2 privileges, or the print file was created on the account which is executing the SP-KILL, and will leave the message ABORT! at the end of the output.
SP-KILL 3-5	This will cause termination of the print files going to printers 3, 4, and 5, subject to the considerations on print file noted above.
SP-KILL B	This will cause termination of the print files going to all the printers, again subject to the considerations above.
SP-KILL 3N	As above, but without the abort message at the end of the report.
SP-KILL (N3	The same as above.
SP-KILL A	Will terminate output from each printer which is outputting a report which was created on the account onto which you are now logged.
SP-KILL A2	Will terminate the output on printer 2 if it is outputting a print file created on the account onto which you are now logged.

Examples of print file termination with the SP-KILL verb.

7.8.2 DEQUEING PRINT FILES.

The SP-KILL verb is also used to deque print files which have been spooled either in the normal course of the generation of the print file, or by means of the SP-EDIT process. Dequeueing is indicated by means of the F option. Print file elements which are to be dequeued are addressed by their print file identification number, which is the number sent to the screen when the print file is initiated, when it is SP-EDITed, and when the LISTPEQS verb is used. They are not identified by the number of the output queue onto which they are linked, which means, among other things, that if something should happen to the links in the output queue chain, the individual elements can be retreived and then reSP-EDITed onto an output queue.

When a print file is enqueued, it is marked as being spooled; and the mark will show up in the LISTPEQS display of the print file control record as an "S". If a print file is not enqueued, it can not be dequeued. When a print file is dequeued, it is set to hold file status and made available for SP-EDITing. A print file which is marked as being output, indicated by an "O" in the LISTPEQS display, will not be dequeued, except under the conditions of the O suboption. The O suboption has the effect of dequeuing print files which are either hold files, or are specified for multiple copies and are not on the last copy, or are open and not choked. In none of these cases has the storage release process started. If the storage release process has started, as in the other cases, the print file will not be dequeued.

When the printer completes output of a print file which has been dequeued, it searches the expected queue, and upon not finding the expected control record, simply tidies up, shrugs, and goes upon its way.

There are assorted reasons for dequeuing print files. They may be enqueued unintentionally, due to an incorrect SP-ASSIGN specification, or mistakenly, into the wrong output queue. If there is more than one printer available, it may be convenient to move some work from a full ouput queue being serviced by one printer to the printer with less work. It may be decided that the report is erroneous in some way, and instead of waiting to execute an SP-KILL when it starts printing, one can dequeue and delete it now. Generally, any change in plans or system protocols may make it advantageous to move print files about. Note that in making decisions as to what print file to move where, there are LISTPEQS facilities, which show the size of each element in the queue, and its account, line, and time of generation, as well as the L option for the SP-EDIT verb, which allows the inspection of members of the output queue.

Note that any print file can be dequeued by an account with SYS2 privileges, but other accounts can only dequeue print files generated on the account onto which they are now logged.

SP-KILL F5	Will dequeue print file 5 if it is enqueued, and if the account executing the verb has SYS2 privileges or is the account which generated the print file.
SP-KILL F5-10	Will dequeue print files 5 through 10, inclusive if they are enqueued, or any which are enqueued, and report the status of each, with the limitations noted above.
SP-KILL FB	Will deque all print files currently enqueued, subject to the ownership conditions noted above.
SP-KILL F7O	Will deque print file 7 even if it is being output, if it is not being released to overflow during output, and subject to the other considerations supra.
SP-KILL (7FO	The same as above.
SP-KILL O5-10F	The same as SP-KILL F5-10, but with dequeuing of print files being output, subject to the usual considerations.
SP-KILL A	Will dequeue all print files created on the account onto which you are now logged.
SP-KILL 5-9A	Will deque all print files with entry numbers 5 through 9 inclusive which were created on the account onto which you are now logged.

Examples of dequeuing with SP-KILL F.

7.8.3 DELETING A PRINTER FROM THE SYSTEM.

The SP-KILL verb with the D option is used to delete a printer from the system. The effect of the action is to clear the printer control block to its initial state, and to clear the task control block used by the printer to its initial state. The printer looses its mind at this point. If it is outputing a file when it is deleted, the print file will remain in never-never land until an SP-KILL D is executed on it. If the printer process was releasing space to overflow when the deletion occured, the rest of the space used by the print file is now lost, and the control record is retreivable only by execution of the :STARTSPOOLER verb. The residual control record is harmless, but tiresome to look at.

If the printer which is deleted is a parallel printer, the effect of the deletion is to cause the spooler to not-activate that printer until it is reinitialized. If the printer was a serial printer, the printer deletion process sends the line occupied by the printer to LOGON, so that it may then be used as a normal communication port.

Note that only accounts with SYS2 privileges may delete printers from the system, and that the A option is not applicable.

SP-KILL D1	Will delete printer 1, which is a parallel printer.
SP-KILL D7	Will delete printer 7, which is a serial printer, and send its process to LOGON.
SP-KILL 3-5D	Will delete printers 3 through 5 inclusive.
SP-KILL BD	Will delete all printers from the system.

Examples of deletion of printers by SP-KILL D.

7.8.4 SP-KILL MESSAGES.

7.8.4.1 General messages.

Erroneous specifications may yield one of the following messages:

ILLEGAL PRINTER NUMBER. MUST BE BETWEEN 0 AND 7 INCLUSIVE.
ILLEGAL SPECIFICATION NUMBER nnn.

An attempt to execute a procedure restricted to SYS2 privileged accounts when logged onto an account with a lower privilege level will yield the following message, and a return to TCL.

YOUR SYSTEM PRIVILEGE LEVEL IS NOT SUFFICIENT FOR THIS STATEMENT.

The occurance of a printer contol block which is in an ambiguous state will yield the following:

PRINTER # n CONTROL BLOCK HAMMERED. CLEARED TO NULL.

This means that the printer cannot be accessed. It should be restarted with the use of the initialization form of the STARTPTR verb, and then either used or deleted. If it is active and printing a real report, it should be allowed to finish. It will then go to sleep. If it is reinitialized while printing a report, that report and its storage will be lost.

7.8.4.2 SP-KILL messages.

The following messages may result from the execution of SP-KILL:

Successful execution of the SP-KILL verb will yield the message:

JOB ABORTED ON PRINTER # n.

Unsuccesful execution will yield the following:

PRINTER # n IS INACTIVE.
THE JOB BEING OUTPUT ON PRINTER # n IS NOT YOUR PRINT FILE.

7.8.4.3 SP-KILL F messages.

Execution of the SP-KILL F verb will yield one of the following messages:

PRINT FILE # nnn WAS NOT UNLINKED BECAUSE IT IS BEING OUTPUT.

PRINT FILE # nnn WAS NOT UNLINKED BECAUSE IT IS UNUSED.

PRINT FILE # nnn WAS NOT UNLINKED BECAUSE IT WAS NOT SPOOLED.

PRINT FILE # nnn WAS NOT CREATED ON THE ACCOUNT ONTO WHICH YOU ARE NOW LOGGED.

Successful dequement is indicated by:

PRINT FILE # nnn WAS UNLINKED AND IS AVAILABLE AS A HOLD FILE.

7.8.4.4 SP-KILL D messages.

Execution of the SP-KILL D verb will yield one of the following messages:

PARALLEL PRINTER # n HAS BEEN DELETED.

SERIAL PRINTER # n HAS BEEN DELETED, AND ITS PROCESS SENT TO LOGON.

7.9 THE LISTPEQS VERB.

The LISTPEQS verb enables the user to interrogate the permanent print file control record area. It allows the system manager and system users to find out the disposition of individual print files, the activity on specific accounts, and the condition of spooler storage.

LISTPEQS will display the relevant information in print file control block elements about both live and recently released print files.

The print file control block is set up when the first line of output from the generating process is 'spooled'. The generating process enters the spooling routine and checks whether it has a place to put the current line. If it does not have the necessary storage structure constructed, it proceeds to obtain a print file control block on a first available basis, and a frame of storage for the line of output. It transfers the output specifications for the job from the SP-ASSIGment block and the print file storage location to the print file control block, as well as the line's identification and the date and time of initiation, and then goes about its business. It returns at close time to mark the control block element and to transfer the number of frames used.

The print file control block element will persist until the print file is deleted from the system by either the spooler or an SP-EDITing process, at which time the print file control block element will be marked available. The element will remain with the terminal condition information until it is used by another print file. How long that may be depends on the relative rate at which print files whose control blocks preceed the control block in question are made available, and the rate at which new print files are generated.

Note that the number of frames used will be spurious in the case of SP-ASSIGN I and IC jobs. The number of frames used will reflect the number of frames left to print when the print file was closed, rather than the total number of frames used. This will be importance only in the SP-ASSIGN HI case, or if multiple copies are desired, and in the case that precise overflow availablity calculations are being performed.

7.9.1 LISTPEQS OPTIONS.

The LISPTEQS verb has several options which allow selectivity of display.

A	Displays only the print files generated on the account on to which you are currently logged.
L	Displays those print files which are deleted as well as those which are active.
C	Displays only the number of print files and the amount of storage used thereby as totals for the class of print files specified by other options.
F	Displays print files enqued for output in their output sequence in each non-null queue by queues in the natural order.
E	Displays print file real storage location. This is to be used in conjunction with the DUMP verb by the instalation manager in moments of stress.
'accountname'	Displays all print files generated on the account with the name 'accountname'.
P	Causes the results of the verb to be printed. P
n	Displays print file control block entry # n, which is the same number output as a message upon the initiation of the print file control block element.
n-m	Displays print files n through m, inclusive, as above.

The LISTPEQS verb's options.

Definition of LISTPEQS FIELDS:

```
 # STAT LK LN STATUSES          CP FO FRMS   DATE     TIME   ACCT
16 C009    11 A (HP C  RL )     1  4    38 01/31/79 16:42:30 TSB
                                                            |-- ACCT
                                                                NAME
                                                      |-- creation time
                                                   |-- creation date
                                                |-- number of frames in
                                                    closed file; OPEN if
                                                    open; residue at close
                                                    for choked file.
                                             |-- form number from SP-ASSIGN or
                                                 SP-EDIT R.
                                          |-- copy count from SP-ASSIGN or
                                              SP-EDIT R.
                                    |-- print file statuses:
                                    ---------------------------------------------
                                    | A   Available          H    Holdfile      |
                                    | P   Printer            T    Tape          |
                                    | I   Immediate          G    aliGn         |
                                    | N   No close           C    Closed        |
                                    | S   Spooled            O    being Output  |
                                    | X   aborted (SP-KILL)  R    Requeued (SP-EDITed)
                                    | L   Locked                                |
                                    ---------------------------------------------
                                 |-- Line number of generation or SP-EDIT spooling.
                              |-- Forward link, if nonzero.
                         |-- Status tally.
                    |-- Spooler permanent entry number for SP-EDIT use.
```

Spooler permanent Q element display explaination.

7.9.3 LISTPEQS STATUS INDICATORS.

Note that if the element status is available, the leading indicator will be an A and that the indicators of the terminal status of the element will be enclosed in parentheses. The other status indicators are grouped into four blocks delimited by blanks.

A (...) specifies that the control block is available.

7.9.3.1 JOB CHARACTERISTICS:

H specifies that this is a hold file, either by virtue of the SP-ASSIGNment under which it was created, or because it was dequeued by the execution of the SP-KILL F verb, or due to the occurance of a cold start.

P specifies that it is to go to the printer either from the initial SP-ASSIGNment, or by virtue of the SP-ASSIGNment in effect at SP-EDIT time. In the latter case the R indicator will probably be on.

I indicates that the print file was linked on to its specified output queue when it was initiated.

N specifies that the print file was generated under an SP-OPEN condition or SP-ASSIGN O specification.

G indicates that the control record references a phantom print file which is the alignment segment of another print file which was aligned.

T indicates that the print file has gone to tape by means of the SP-EDIT process.

7.9.3.2 CLOSED CONDITION:

C indicates that the print file is closed; its absence indicates that the print file is still open, which means that it is currently being generated, or it is in an SP-OPEN condition.

7.9.3.3 ENQUEUED CONDITION:

S indicates that the print file is linked on to an output queue, and as such should appear in an execution of LISTPEQS F. If it does not, execute an SP-KILL Fn, and then reenque it as necessary.

O indicates that the print file is being output. As such, it should be spooled and have that condition indicated by the S indicator above. Further, it should show up in the LISTPEQS F display, and as the print file being output by one of the printers as displayed by the SP-STATUS verb. If the print file has disappeared from the LISPTEQS F display but not the SP-STATUS display, it should terminate normally; if it has disappeared from the SP-STATUS display, an SP-KILL FOn is probably called for. If that does not remove the control record, a :STARTSPOOLER will. Be sure to check the record twice, however, because it is possible to display the record during transitions.

7.9.3.4 SP-EDIT conditions:

L indicates that the record is locked. It can not be SP-EDITed when it is locked, except by the SP-EDIT L option. The control block record will be locked when it is available, open, spooled, being output, or while being SP-EDITed. Only one process can deal with a print file at a time. It should return to an unlocked state after any abnormal termination of the SP-EDIT process. It will be set to hold file status and unlocked at cold start time, unless it does not pass certain validity tests.

R indicates that the print file has been SP-EDITed and sent to the printer at some time.

X indicates that the print file has been aborted by the SP-KILL process at some time.

7.9.4 Examples of the LISTPEQS verb.

```
>LISTPEQS L

    PRINTER LIST ELEMENTS                    02 FEB 1979 12:51:05

  # STAT LK LN STATUSES          CP FO FRMS   DATE     TIME    ACCT

  1 C080    11 HP C              1  4  261 01/31/79 17:49:32 TSB
  2 C080     1 HP C              1  4  575 02/02/79 10:58:54 TSB
  3 81C0     5 H C               1  0   37 02/01/79 10:37:52 MANUALS
  4 C088     0 HP C  R           1  0   81 02/01/79 10:32:53 BUGEYE
  5 81L1     5 H  C              3  0 OPEN 02/02/79 12:51:06 DP
  6 41C1     7 P  L              1  0 OPEN 02/02/79 12:43:06 MANUALS
  7 8001     5 A (H C  L )       3  0    2 02/02/79 12:44:18 DP
  8 C088     9 HP C  R           1  0   30 02/01/79 09:43:19 BUGEYE
  9 C098     5 HP C  XR          5  1   21 01/31/79 12:06:56 DP
 10 C080     1 HP C              1  4  420 02/01/79 09:48:16 TSB
 11 8001     5 A (H C  L )       3  0    1 02/02/79 12:40:05 DP
 12 6001     1 A (PI C  L )      1  0    5 01/31/79 18:44:49 BFS
 13 C100     5 A (HP C  )        1  5    1 01/31/79 15:38:59 CAROL
 14 C019    11 A (HP C  XRL )    1  0    2 01/31/79 16:04:21 TSB
 15 C098     7 HP C  XR          1  0   19 01/31/79 16:12:08 CAROL
 16 C009    11 A (HP C  RL )     1  4   38 01/31/79 16:42:30 TSB
 17 4011     3 A (P C  XL )      1  0   64 01/31/79 17:09:12 ADM
 18 C001    11 A (HP C  L )      1  4    1 01/31/79 17:10:21 TSB
 19 4001     7 A (P C  L )       1  0    5 01/31/79 17:10:59 CAROL
 20 4001     7 A (P C  L )       1  0    3 01/31/79 17:11:07 CAROL
 21 4001     7 A (P C  L )       1  0   17 01/31/79 17:11:16 CAROL
 22 4001     7 A (P C  L )       1  0   22 01/31/79 17:11:30 CAROL

 22 QUEUE ELEMENTS LISTED.              1609 FRAMES IN USE.
```

Listing of all permanent Q elements.

```
>LISTPEQS AL

    PRINTER LIST ELEMENTS                    02 FEB 1979 12:51:17

  # STAT LK LN STATUSES          CP FO FRMS   DATE     TIME    ACCT

  5 8080     5 H C               3  0    4 02/02/79 12:51:06 DP
  7 80C1     5 H  L              3  0 OPEN 02/02/79 12:51:18 DP
  9 C098     5 HP C  XR          5  1   21 01/31/79 12:06:56 DP
 11 8001     5 A (H C  L )       3  0    1 02/02/79 12:40:05 DP

 4 QUEUE ELEMENTS LISTED.                26 FRAMES IN USE.
```

Listing of control records of print files generated on this account.

```
>LISTPEQS

    PRINTER LIST ELEMENTS                  02 FEB 1979 12:51:55

  # STAT LK LN STATUSES          CP FO FRMS   DATE     TIME   ACCT

  1 C080    11 HP C              1  4  261 01/31/79 17:49:32 TSB
  2 C080     1 HP C              1  4  575 02/02/79 10:58:54 TSB
  3 81C0     5 H C               1  0   37 02/01/79 10:37:52 MANUALS
  4 C088     0 HP C  R           1  0   81 02/01/79 10:32:53 BUGEYE
  5 8080     5 H C               3  0    4 02/02/79 12:51:06 DP
  6 41C1     5 P  L              1  0 OPEN 02/02/79 12:43:06 MANUALS
  7 8080     5 H C               3  0    1 02/02/79 12:51:18 DP
  8 C088     9 HP C  R           1  0   30 02/01/79 09:43:19 BUGEYE
  9 C098     5 HP C  XR          5  1   21 01/31/79 12:06:56 DP
 10 C080     1 HP C              1  4  420 02/01/79 09:48:16 TSB
 11 80C1     5 H  L              3  0 OPEN 02/02/79 12:51:56 DP
 15 C098     7 HP C  XR          1  0   19 01/31/79 16:12:08 CAROL

 12 QUEUE ELEMENTS LISTED.              1449 FRAMES IN USE.
```

Listing of Q elements retaining storage.

```
>LISTPEQS 5-10

    PRINTER LIST ELEMENTS                  02 FEB 1979 12:52:43

  # STAT LK LN STATUSES          CP FO FRMS   DATE     TIME   ACCT

  5 8080     5 H C               3  0    4 02/02/79 12:51:06 DP
  6 41C1     5 P  L              1  0 OPEN 02/02/79 12:43:06 MANUALS
  7 8080     5 H C               3  0    1 02/02/79 12:51:18 DP
  8 C088     9 HP C  R           1  0   30 02/01/79 09:43:19 BUGEYE
  9 C098     5 HP C  XR          5  1   21 01/31/79 12:06:56 DP
 10 C080     1 HP C              1  4  420 02/01/79 09:48:16 TSB

 6 QUEUE ELEMENTS LISTED.               476 FRAMES IN USE.
```

Listing of Q elements 5 through 10.

```
>LISTPEQS C

    PRINTER LIST ELEMENTS                  02 FEB 1979 12:53:37

 12 QUEUE ELEMENTS LISTED.              1449 FRAMES IN USE.
```

Count number of live Q elements and the number of frames used.

```
>LISTPEQS F

 FORM QUEUE    0

     PRINTER LIST ELEMENTS                15 FEB 1979 12:08:37

    #  STAT  LK LN STATUSES          CP FO FRMS   DATE     TIME   ACCT

    3 4085   4  3 P C S L            1  0   77 02/15/79 11:09:48 ADM
    4 4085   8 13 P C S L            1  0    1 02/15/79 11:32:40 JB
    8 4085     14 P C S L            1  0    2 02/15/79 11:52:20 CHRIS

 FORM QUEUE    4

    5 C0AD      0 HP C SO RL         1  4  250 02/15/79 10:26:43 DP

 FORM QUEUE    5

    6 4085   2  0 P C S L            1  5    1 02/15/79 11:52:09 DP
    2 C09D      0 HP C S XRL         3  5  420 02/14/79 22:08:31 TSB

    6 QUEUE ELEMENTS LISTED.              751 FRAMES IN USE.
```

Listing of Q elements by output Q

In this case, note that there is a printer outputing print file control block entry 5, which is in form queue 4.

```
>LISTPEQS 'MANUALS'

     PRINTER LIST ELEMENTS                11 MAY 1979 11:55:13

    #  STAT  LK LN STATUSES          CP FO FRMS   DATE     TIME   ACCT

    1 C8AD   7  7 HP C SO RL          1  0 3946 05/10/79 10:57:15 MANUALS
    2 C888      7 HP C  R             1  0   67 05/10/79 11:32:26 MANUALS
    4 8880      0 H C                 1  0 3945 05/11/79 10:42:48 MANUALS
    5 8880      0 H C                 1  0   67 05/11/79 11:02:44 MANUALS
    6 8880      0 H C                 1  0  107 05/11/79 11:05:33 MANUALS
    7 C88D      7 HP C S RL           1  0  128 05/08/79 08:00:48 MANUALS

    6 QUEUE ELEMENTS.                    8260 FRAMES IN USE.
```

Example of the use of LISTPEQS 'accountname'.

Note that if other options are used, they must be either in from of the account name, or preceeded by a left parenthesis, as in the normal system options case. Also, the account name must preceed any left parenthesis, as with all other processors on the system.

In this particular example, print file # 1 is being output, and print file # 7 is linked on behind it.

7.10 THE LISTPTR VERB.

The LISTPTR verb lists the currently allocated printer control blocks, including number, status, and forms allocated.

The use of the LISTPTR verb is to discover the state of the printers and their form allocations. It is to be used in conjunction with the STARTPTR, LISTPEQS, and SP-EDIT verbs, and when the information given by the SP-STATUS verb is more than you want for the moment.

```
PRINTER          --   FORMS   --    PAGE  DEV OR     STATUS
TYPE   NUMBER                       SKIP  LINE #

PARALLEL   1      0     4             0       1      INACTIVE
|          |     |-----------|       |       |      |
|          |                         |       |      |-- Status
|          |                         |       |          explanation.
|          |                         |       |
|          |                         |       |-- Parallel printer
|          |                         |           number or line
|          |                         |           number.
|          |                         |
|          |                         |-- Number of pages to skip
|          |                             between printfiles.
|          |
|          |-- Numbers of form queues being processed
|              by this printer.
|
|-- Printer number -- for STARTPTR, SP-KILL, STOPPTR.
|
|-- Printer type -- parallel or serial.
```

Definition of LISTPTR fields.

STOPPED	The printer is set to stop.
INACTIVE	The printer is inactive. If it is also STOPPED, the STARTPTR verb may be used on the printer.
ACTIVE	The printer is printing a report, or intitating or terminating a print file.
UNALLOCATED	The printer has never been started, or has been deleted by the SP-KILL D verb, or has been lost due to a control block error. It may be started by the STARTPTR verb.

LISTPTR statuses.

```
>LISTPTR

PRINTER ASSIGNMENTS                                        12:40:06

PRINTER          --   FORMS   --   PAGE  DEV OR    STATUS
TYPE   NUMBER                      SKIP  LINE #

PARALLEL   0      1                  0       0    INACTIVE STOPPED
PARALLEL   1      0      4           0       1    INACTIVE
```

Normal form of the LISTPTR verb.

```
>LISTPTR 1

PRINTER ASSIGNMENTS                                        12:44:11

PRINTER          --   FORMS   --   PAGE  DEV OR    STATUS
TYPE   NUMBER                      SKIP  LINE #

PARALLEL   1      0      4           0       1    INACTIVE
```

Limited form of the LISTPTR verb.

```
>LISTPTR

PRINTER ASSIGNMENTS                                        14:16:35

PRINTER          --   FORMS   --   PAGE  DEV OR    STATUS
TYPE   NUMBER                      SKIP  LINE #

PARALLEL   0      0                  0       0    INACTIVE
SERIAL     7     11                  3      11    INACTIVE
```

LISTPTR with a serial printer allocated.

```
>LISTPTR B

PRINTER ASSIGNMENTS                                        12:44:19

PRINTER          --   FORMS    --    PAGE  DEV OR     STATUS
TYPE   NUMBER                         SKIP  LINE #

PARALLEL   0      1                     0        0     INACTIVE
PARALLEL   1      0      4              0        1     INACTIVE
SERIAL     2                                          UNALLOCATED
SERIAL     3                                          UNALLOCATED
SERIAL     4                                          UNALLOCATED
SERIAL     5                                          UNALLOCATED
SERIAL     6                                          UNALLOCATED
SERIAL     7                                          UNALLOCATED
```

Inclusive form of the LISTPTR verb.

Error message number	Meaning
1171	The printer is inactive.
1172	The printer is active.
1174	The printer is unallocated.

LISTPTR error message numbers.

Note from the examples that it is possible for a printer to be STOPPED and either ACTIVE or INACTIVE. This is becuase the STOPPTR verb only marks the printer to stop at the completion of the current job or copy of a print file. A printer may not be restarted until the printer is both INACTIVE and STOPPED.

The LISTPTR verb generates the following pattern in the PROC secondary input buffer: The first argument will always be 1134, which is the source of the heading in the display. Each printer is then represented as a block of either two or six arguments. If the printer is unallocated, there will be two arguments; if the printer is allocated, there will be six. The first element in each printer block is a condition value, as in the table above. Printers with the condition 1174, unallocated, will have one argument following the condition code. Printers with the condition 1171, inactive, and 1172, active, will have five arguments following the condition code. The argument following the condition code is the printer number in all cases. This finishes the block for unallocated printers.

If the printer is allocated, the printer number will be followed, in order, by the number of the first-specified output queue, the number of the second-specified output queue, and the number of the third-specified output queue. The sixth argument in the block is the number of pages to eject.

Below is a picture and an example, with the PROC secondary input buffer contents which result from the execution of the verb in a PROC.

```
>LISTPTR 1

PRINTER ASSIGNMENTS                                    12:44:11

PRINTER        --   FORMS   --   PAGE  DEV OR    STATUS
TYPE   NUMBER                    SKIP  LINE #

PARALLEL   1     0     4            0       1    INACTIVE
          /     /     /     /       /             /
         --+-----+-----+-----+-------+-------------
           |     |    /     /       /
           |     |   /  ----       /
           |     |  /      ----    /
           |     | /          ---------
           |     |/
1134 1171  1  0  4  127  0
  |        |     |     |  |
  |        |     |     |  -- page skip specification.
  |        |     |     |
  |        |     |     -- queue queue 3; 127 specifies no queue
  |        |     |        and is internal flag.
  |        |     |
  |        |     -- output queue 2; in this case, 0.
  |        |
  |        -- output queue 1; in this case, 1.
  |
  |     -- printer number 1.
  |
  |     -- the printer is inactive.
  |
  -- the buffer contents are marked as a LISTPTR result.
```

LISTPTR and the PROC secondary input buffer: one printer.


```
>LISTPTR 0-4

PRINTER ASSIGNMENTS                                        12:44:19

PRINTER          --   FORMS    --    PAGE  DEV OR     STATUS
TYPE   NUMBER                         SKIP  LINE #

PARALLEL    0      1                    0        0     ACTIVE STOPPED
PARALLEL    1      0      4             0        1     INACTIVE
SERIAL      2                                         UNALLOCATED
SERIAL      3                                         UNALLOCATED
SERIAL      4                                         UNALLOCATED


1134 1172 0 1 127 127 0 1171 1 0 4 127 0 1174 2 1174 3 1174 4
   |   |                |  |               |  |    |  |    |  |    |
   |   ----------------- ---------------- ------ ------ ------
   |                    |                |      |        |
   |                    |                |      |        |
   |                    |                |      |        -- printer
   |                    |                |      |           4 block
   |                    |                |      |
   |                    |                |      -- printer 3 block
   |                    |                |
   |                    |                -- printer 2 block.
   |                    |
   |                    -- printer 1 block; inactive.
   |
   |                -- printer 0 block; active.
   |
   -- the LISTPTR PROC secondary input buffer residue mark.
```

LISTPTR PROC secondary input buffer contents after listing several printers.

In this example, note that allocated printers have six arguments, that unallocated printers have two, that the first argument in each block is the printer condition code, and that the second is the printer number. The third through sixth arguments are the output queue and page skip specifications. The output queue value of 127 specifies that there is no output queue specified for that element.

7.11 THE LISTABS VERB.

The LISTABS verb lists the current assignment of all the lines on the system.

The LISTABS verb is useful if there is a question as to the allocation of a running process, or if the system manager wishes to discover why print files seem to be going unexpected places.

```
>LISTABS

  LINE   STATUS  COPIES  FORM #
     #
     0   P          1       5
     1   PI         1       0
     2   P          4       0
     3   PIC        1       0
     4   PI         1       0
     5   PO         1       0
     6   HT         1       0
     7   H          1       0
     8              0       0
     9              0       0
    10              0       0
    11              0       0
    12              0       0
```

Example of the LISTABS verb.

P	Print output.
H	Create and keep a holdfile copy. H
T	Output to tape.
I	Enque the job at the beginning of the job.
C	Choke the creating process to printer speed.
O	Keep the print file open at close time.

Status indicators for the LISTABS verb.

The SP-STATUS verb displays the current status of the spooler system and of each printer defined for the system. The SP-STATUS verb also has the effect of awakening the spooler if he was asleep.

7.12.1 THE SP-STATUS VERB AS A SYSTEM INFORMATION DISPLAY

The intent of the SP-STATUS verb is to give a general over-view of the system. The message will indicate whether the spooler is active or inactive. Activity normally implies that one or more of the parallel printers which run as subtasks of the spooler are attempting to ouput. The spooler may be executing one of its administrative tasks, however. There are other messages indicating that request flags have been set by other processes. The persistance of such messages may indicate that all is not well with the spooler, and measures may have to be taken.

The SP-STATUS message process will then report on each possible printer. It will specify the type and status of each printer, its form and page skip specifications; and if the printer is active, it will note the print file element number, line number and account name of the generating process, and its size if closed.

7.12.2 THE SP-STATUS VERB AS SPOOLER AWAKENER.

All processes which request activity from the spooler will awaken it. There may be cases in which it continues to snooze, however, in which case the use of the SP-STATUS verb will awaken the spooler and cause it to go look for work to do. Therefore, if it appears that the spooler is neglecting a task, it may be encouraged by the use of the SP-STATUS verb. If this does not have the desired effect, then various other approaches should be taken. For instance, check to see if the printer is on line, that the printer is allocated to the intended output queue, and that the desired print file exists, is enqueued and is enqueued in the correct output queue. The side-effect of the SP-STATUS verb awakening the spooler is that the messages generated by the verb may be transiently spurious because the spooler is looking for work, and in the case that the system load is relatively constant, several executions of the SP-STATUS message may encounter exactly the same transient condition several times in succession without actually finding an error condition, even though it is reported. Further investigation is recommended.

7.12.3 THE ON-LINE AND OFF-LINE CONDITION.

Note that the ON LINE and OFF LINE are often imprecise in the following ways and for the following reasons. If the printer is inactive, it necessarily had to complete the last job successfully, at which time the printer was on line, and the SP-STATUS message so indicates whether the physical printer is on line or off line. There is no specific check for the current condition of the physical printer if the logical printer is inactive for two reasons. First, only the spooler process attends to the

parallel printers, and only the serial printer process can attend to its physical serial printer. Second, since the parallel printer subprocess initializes the printing of a block of text when the printer is "on line", and then loops while the transfer is taking place, the normal condition seen from an active physical parallel printer is "not ready" or "off line" because it is currently printing a block of text and is not ready for the next block. The result of this is that if the SP-STATUSing process checks the physical printer while it is printing, it will normally appear to be off line when it is either on line or off line. Therefore, the SP-STATUSing process checks only what the printer output control block thinks the status of the printer is, since, given the wires available, it is not analytical. If it is printing, then the messages carried by the STATUSing are not passed. In that case, they are reserved for the WHERE section of the WHAT verb. The P and B options will still be effective. The P option will print all of the WHAT verb's results, a facility which is possibily of use in documenting system difficulties.

none	The status of the spooler and all allocated printers will be displayed.
B	The status of the spooler and all printers, allocated and unallocated, will be displayed.
P	The SP-STATUS results will be printed.
n	The status of the spooler and the status of printer will be displayed. The option n must be between 0 and 7, inclusive.
n-m	The status of the spooler and the status of printers n through m will be displayed. The option m must be between n and 7, inclusive.

SP-STATUS options.

```
>SP-STATUS B


THE SPOOLER IS ACTIVE.

PRINTER # 0 IS PARALLEL, ACTIVE, AND ON LINE.
THE PRINTER IS DEFINED AS PARALLEL PRINTER # 0.
PRINT FILE BEING OUTPUT IS ELEMENT 8, A CLOSED FILE FOR LINE # 6
  GENERATED ON ACCOUNT DP, WHICH IS 13 FRAMES LONG.
ASSIGNED OUTPUT QUEUES: 0.
THE NUMBER OF INTER-JOB PAGES TO EJECT IS 0.

PRINTER # 1 IS UNALLOCATED.
PRINTER # 2 IS UNALLOCATED.
PRINTER # 3 IS UNALLOCATED.
PRINTER # 4 IS UNALLOCATED.
PRINTER # 5 IS UNALLOCATED.
PRINTER # 6 IS UNALLOCATED.
PRINTER # 7 IS UNALLOCATED.
```

Example of the SP-STATUS verb with all printers displayed.

```
>SP-STATUS


THE SPOOLER IS ACTIVE.

PRINTER # 0 IS PARALLEL, ACTIVE, AND ON LINE.
THE PRINTER IS DEFINED AS PARALLEL PRINTER # 0.
PRINT FILE BEING OUTPUT IS ELEMENT 8, A CLOSED FILE FOR LINE # 6
  GENERATED ON ACCOUNT DP, WHICH IS 13 FRAMES LONG.
ASSIGNED OUTPUT QUEUES: 0.
THE NUMBER OF INTER-JOB PAGES TO EJECT IS 0.
```

Normal form of the SP-STATUS display.

```
>SP-STATUS 4-7

THE SPOOLER IS INACTIVE.

PRINTER # 7 IS SERIAL, INACTIVE, AND ON LINE.
THE PRINTER IS RUNNING ON LINE 11.
ASSIGNED OUTPUT QUEUES: 8, 9.
```

Example of SP-STATUS with a range and an allocated printer.


```
>SP-STATUS

THE SPOOLER IS INACTIVE.

PRINTER # 0 IS PARALLEL, INACTIVE, AND ON LINE.
THE PRINTER IS DEFINED AS PARALLEL PRINTER # 0.
ASSIGNED OUTPUT QUEUES: 0.
THE NUMBER OF INTER-JOB PAGES TO EJECT IS 0.

PRINTER # 7 IS SERIAL, INACTIVE, AND ON LINE.
THE PRINTER IS RUNNING ON LINE 11.
ASSIGNED OUTPUT QUEUES: 8, 9.
THE NUMBER OF INTER-JOB PAGES TO EJECT IS 3.
```

Example of SP-STATUS with a serial printer allocated.

```
>SP-STATUS 0-2B

THE SPOOLER IS INACTIVE.
PRINTER # 0 IS UNALLOCATED.
PRINTER # 1 IS UNALLOCATED.


THE CONTROL BLOCK FOR PRINTER # 2 IS IN AN AMBIGUOUS STATE.
DELETE THE PRINTER FROM THE SPOOLER SYSTEM.
```

SP-STATUS with an erroneous transient.

The example above was taken precisely when printer 2 was checking to see if there was more work for it to do. There are various transient cases when the condition of the spooler may be reported to be strange. When the timings on the system change due to changed work loads, the transients will become invisible, and the message will become normal.

On the other hand, the message may be telling the truth, in which case use of the LISTPTR verb, using the printer as the destination of a test print file, or using the STOPPTR verb should generate an irregularity, in which case the use of the SP-KILL D verb, followed by the use of the full STARTPTR verb is called for. Should this not prove sufficient, see the discussion of the :STARTSPOOLER verb.

Error message	Number
THE SPOOLER IS INACTIVE.	1200
THE SPOOLER IS ACTIVE.	1201
NEEDS TO START PRINTERS.	1202
NEEDS TO LOG DISC ERRORS.	1203

SPOOLER condition error messages and numbers.

The messages 1200 and 1201 refer to the spooler process rather than a single printer. If it is INACTIVE, it is asleep and using no CPU resources; if it is ACTIVE, it is operating one or more of the parallel printers, starting printers, or logging disk errors. Messages 1202 and 1203 refer to flags set by other processes to cause action on the part of the spooler process. The spooler is flagged to start printers and awakened when a print file is enqueued, when a printer is started by the STARTPTR verb,and under certain options of the :STARTSPOOLER verb. The spooler is flagged to log disk errors and is awakened if a process encounters a disc error. The spooler is also awakened by the SP-STATUS verb.

If messages 1202 and 1203 persist, there may be problems with the system. Normally the spooler process turns the flag off when it executes the necessary activity. If the flag which causes message 1202 stays on, it is probably because the spooler is probably confused, and various forms of :STARTSPOOLER should be executed, although there are cases of timing where the message will be fairly persistent when there is no problem. If message 1203 persists, either the spooler is confused, as above, or the system has a hard disk error. Check the tenth section of this manual for disk error log retrieval.

Error message	Number
THE CONTROL BLOCK FOR PRINTER # n IS IN AN AMBIGUOUS STATE. DELETE THE PRINTER FROM THE SPOOLER SYSTEM.	1209
PRINTER # n IS	1210
UNALLOCATED	1211
SERIAL	1212
PARALLEL	1213
INACTIVE	1214
ACTIVE	1215
STOPPED	1216
AND ON LINE.	1217
AND OFF LINE.	1218
THE PRINTER CABLE IS OFF	1219
THERE IS NO CONTROLLER FOR THIS PRINTER	1220
THE PRINTER IS DEFINED AS PARALLEL PRINTER # n.	1221
THE PRINTER IS RUNNING ON LINE n.	1222
PRINT FILE BEING OUTPUT IS ELEMENT n	1229
AN OPEN FILE FOR LINE # n	1230
A CLOSED FILE FOR LINE # n	1231
GENERATED ON ACCOUNT accountname	1232
WHICH IS n FRAMES LONG.	1233
AND THE OUTPUT IS CHOKED.	1234
ERRONEOUSLY, THE PRINTER HAS NO OUTPUT QUEUES ASSIGNED TO IT.	1239
ASSIGNED OUTPUT QUEUES n,n,n	1240
THE NUMBER OF INTER-JOB PAGES TO EJECT IS n.	1243

PRINTER error messages and numbers.

Note that there are several other error message numbers which may occur. They have to do with conditional punctuation, and are trivial. Note also that, unlike the LISTPTR verb, the numeric arguments which are inserted into the text here are not sent to the PROC secondary output buffer.

Error message 1209 usually means what it says, but it is possible to obtain the error if the SP-STATUS verb is executed at just the wrong time. Reexecute the verb to see if the message persists, and, if it does, see the note under the example above.

Error message 1219 means what it says: If it occurs, there is a discontinuity in the cable between the printer controller and the printer.

Error message 1220 also means what it says. If a parallel printer is started at a device address which is either not in the computer, or is defective, then the message will occur. If the parallel printer was started on a device address which is not there, use the SP-KILL D, and start the correct printer. No harm has been done. If the printer is supposed to operate from that address and does not, call hardware.

The argument in error message 1221 will be the same as the argument in error message 1210.

Error message 1222 refers to serial printers, and the argument will be the line, channel, or port number, as in the WHAT verb.

Error message 1232 gives the account on which the print file was generated, if there is a print file being output. Error messages 1229 through 1234 are applicable only to active printers. Error message 1234 refers to print files generated under SP-ASSIGN CI, and which have the C indicatior under a LISTPEQS display.

Error message 1239 indicates that something is amiss. Stop the printer and restart it, or delete it and restart it. If there is still a problem, a COLDSTART or a patch is probably in order.

Error messages 1240 through 1243 have to do with the display of the form numbers to which the printer is allocated, and its inter-job page eject specification.

7.13 THE COLDSTART AND THE :STARTSPOOLER VERB.

The coldstart process automatically starts the spooler. The :STARTSPOOLER verb is available to restart the spooler if necessary and advisable.

7.13.1 COLDSTART INITIALIZATION OF THE SPOOLER.

The spooler is started during the coldstart process. This occurs immediately after the ABS section has been loaded. There are two paths which may be taken at this point. If the coldstart was executed with a C option, then the coldstarting process will start the spooler, link up its own work space and execute the coldstart proc. If the coldstart was executed with the F option, then the coldstarting process will start the spooler and then execute the file restore.

In both cases the spooler will reinitialize its local constants and its local workspaces. In the case of a C-level coldstart, the spooler will clear all control blocks except the permanent print file control record block. It will attempt to identify all good print files, unlink them from any output chains, make them SP-EDITable, and set its end marker to one after the last good hold file found. If the spooler finds what it considers to be irredeemable garbage, the list of print file control records will be terminated at that point. Note that the effect of this is to clear all printers, output queues, input pointers, and assignments. The spooler then proceeds to link up workspace for all of the lines on the system.

In the case of the F-level cold start, all of the spooler's control blocks are initialized. Any print files which may have been there are gone; and all spare storage is returned to overflow.

7.13.2 THE :STARTSPOOLER VERB'S ACTION.

The :STARTSPOOLER verb allows the execution of these processes selectively and without coldstarting. The minimal execution of the :STARTSPOOLER verb, without any options, is to reinitialize certain global pointers and control data and to send the spooler to a normal sleep. Generally, this will be necessary only if the spooler goes on an unexpected trip to never-never land, which should not happen in any case. The next level is the C option, which clears all the control blocks except the permanent print file control record area. This should not be executed when any spooler-related tasks are live, either generating or printing print files. The global level is the I option, which reinitializes everything the way the F-level coldstart does, except that the storage contained in any extant print files is lost until the next file restore. There is also the L option, which causes the spooler to link up the extended work spaces for all lines which are not logged on at the time. In this case the spooler will also execute a minimal initialization of its control data.

Note especially that the :STARTSPOOLER with the C option should be executed when no processes are generating print files, that all processes will have to reexecute the SP-ASSIGN verb before any further output is attempted, and the the printers must be initialized.

7.13.3 WHEN TO USE THE :STARTSPOOLER VERB.

In general, the :STARTSPOOLER verb should not be used until all other resources have been exhausted, and then only if the spooler is asleep or in the debugger. The spooler may not be responding to a request to print because it is logging disk errors, which is a priority task, for instance. If the printer is not printing when it appears that it should, first try some of the following approaches. Make sure that the printer is on line. This is non-trivial: cold starts have been done because the printer was not on line. Make sure that something is enqueued to print by using the LISTPEQS verb with the F option, and by checking that the printer is allocated to the output queue with available work using the LISTPTR and SP-STATUS verbs. Print files which are available from the spooler's point of view to be printed are enqueued in one of the output queues assigned to the particular printer and are not flagged as being output.

If all of this appears to be in order, and the spooler has been encouraged to attempt output by the execution of the SP-STATUS verb, then the printer is possibly assigned to the incorrect hardware address. This might occur after service, or due to an incorrect execution of the STARTPTR verb. Normally the SP-STATUS verb will point out that the printer controller or printer cable is missing. In this case, delete the printer and reexecute the STARTPTR verb. If all appears good here, it may be that the printer controller board has failed. This happens very seldom, and is probably not the case now.

If there is no progress thus far, create a small, well-known hold file. Inspect it using the SP-EDIT verb. Enqueue it onto an output queue with no other occupants. Start the printer on this output queue only. If nothing happens, that is, nothing is printed, the print file stays enqueued, it is not marked as being output, and it does not appear in the SP-STATUS message, then we must inspect the spooler process to see if it is confused.

Inspection of the spooler process starts with the execution of the WHERE verb, which is in SYSPROG. The spooler is conventionally the last line on the system, so that if you have a 20 line system, the spooler is line 20, the twenty-first process. We are concerned with two things. First, we wish to know if the spooler has an abnormal status; second, we wish to know where it is processing, and if it is processing.

The spooler will normally have one of three statuses, 3F, 5F, or 7F. If the status is 3F, it is either sleeping between jobs, or napping while the printer is outputting a block of data. He will also nap between status-checks to discover whether there is a printer there, and, if so, whether it is on line. All snoozing will be executed in frame 170. If the status is 5F, the spooler is waiting for disk access. Normally this will be observed when it is releasing storage used by an SP-ASSIGN I print file which was open when output was started. In a heavily loaded system this may occur while getting the next frame of print file to output. If the status is 7F, the spooler is enqueued for the CPU. This is a rare occurance.

Abnormal status is normally a 7D40 or a 7B40. These indicate that the spooler is in the debugger and trying to talk to a non-existant terminal. Use the PEEK verb to see what it is trying to say. The spooler will normally be executing in frame 1 at this point.

The spooler will get into the debugger for the following sorts of reasons: 1) The code which the spooler runs in has been modified; 2) the control data which the spooler uses is erroreous, or 3) the spooler has encountered an illegal forward link. In the first case, execute the VERIFY-SYSTEM verb to see if a coldstart is required. If the system does not verify, this may be the case. Check the second case by executing the LISTPEQS, LISTPTR, LISTABS and SP-STATUS verbs in order to ascertain what might be amiss. What looks right and that which looks wrong using these verbs is a matter of practice. If they work properly, there is probably not a problem here. The spooler is primarily concerned with its internal printer control blocks which will be best displayed by the SP-STATUS verb, and which can in general be cleared by the SP-KILL D verb and reinitialized, as has been done above.

This leaves the case of the illegal forward link. It is normally to a frame beyond the range of disk storage on the system, becuase a forward link of zero is the normal mode of termination of a print file. It is the case that in general the system will burp ocasionally. If there is not a recent history of similar spooler problems, catalogued production basic programs which seem to degrade over time, group format errors which reappear, or chains which seem to cross, then the best course is to execute the :STARTSPOOLER verb with no options. It is relatively inocuous and will get things started again. If they stop suddenly again, SP-KILL the print file which the spooler is attempting to print. It probably has a bad link. Deleting the print file will probably result in an ILLEGAL FID message being returned to the deleting process, in which case, END. If the spooler attempts to return the storage, it will retire to the debugger and need to be restarted. Normally, the :STARTSPOOLER verb without options will discard print files which were being output concurrent with storage release because their storage retention is unknown.

If the system has the syndrome of symptoms noted above, depending on the use of the machine, then it is an indication that the computer or its environment is degrading, and it is time to call field service.

If for some reason the control block area of the spooler is a disaster, but the system verifies (there is no known case of this), then execute the :STARTSPOOLER with the C option. If the permanent print file control record area has apparant trash in it which is found to be admissable by the :STARTSPOOLER C, improbable but possible, then delete all print files whose storage is reasonably trust-worthy, and execute the :STARTSPOOLER with the I option.

Print files which print garbage probably have a dubious link some where along the line. Avoid deleting them because, if the print file has a link pointing into valid data, then, when it is deleted, the data area becomes part of the overflow area and group format errors are likely to occur. This is not the only way to print garbage, however.

In general, the :STARTSPOOLER verb need be used only when the spooler has gone off to the debugger due to an erroneous forward link in the print file. Note these occurances. They are a very good indicator of the general health of the system.

:STARTSPOOLER	Initializes control data, and take a normal path to sleep if there is nothing to do.
:STARTSPOOLER C	Initializes control data and all control blocks except the permanent print file control record block. All lines must be reassigned and all printers redefined.
:STARTSPOOLER I	Initializes all control blocks and control data. All lines must be reassigned and all printers redefined.
:STARTSPOOLER L	Causes the spooler to link up the extended work space for all lines not logged on at the time.

The STARTSPOOLER verb and its options.

Note that the C option is a subset of the I option, and that the L option may be used with either or alone if workspace linkage is desired. Note the LINK-WS verb. Note also that the spooler may be started on a line other than the last in the system if a numerical argument is given in the option string used with the :STARTSPOOLER verb.

7.14 SPOOLER VERB OPTIONS HANDLER

Many of the verbs in the spooler process share a common options handler and therefore share the same option handling protocols.

The following verbs share the options handler:

SP-EDIT	SP-KILL	STOPPTR	:STARTSPOOLER
LISTPEQS	LISTPTR	LISTABS	SP-STATUS
T-ATT	T-DET	T-FWD	T-BCK
T-RDLBL	T-READ	SP-TAPEOUT	DUMP
WHO	WHAT	WHERE WHO	

All of the options specified for these verbs could be placed in parentheses as per the normal form for the system. Since there are no file specifications used with these verbs, it is not necessary to use the parentheses. The following rules hold:

It is possible to place some of the options within parentheses and others prior to the parentheses, as long as there is no numeric argument within the parentheses. If there is a numeric argument in the string, then either it must preceed the parentheses, or all of the options must be to the right of a left parentheses. Further, if dual numeric arguments are used, they must have the form n-m with nothing but the hyphen between them. If they are otherwise separated, then the options handler will return the last numeric found as a single parameter. If two numbers are used in the proper way, they return a range, which is taken to be inclusive in all cases where it has meaning. In general, each processor checks the numbers for legality, and sets up a default value for each where possible. In the table below, the defaults are indicated by DN and DM.

If there is more than one number or legal pair of numbers in the option string then the last number or number pair will be retained. If there is a legal number pair, n'-m', early in the string which is loaded into N and M, such that N = n' and M = m', and there is a single number n'' later in the string, the options handler will return N = n'' and M = n''. Similarly, if there is a second legal number pair, n''-m'', in the string after a first pair, n'-m', then the options handler will return N = n'' and M = m''. Do not, therefore, put more than one numeric element, n or n-m, in an options string.

The 'accountname' parameter used with the SP-EDIT and LISTPEQS verbs must be surrounded by ', ", or \ as per the item specification rules of ACCESS. If there are other options specified, they must either be to the right of the left parenthesis, or proceed the 'accountname' specification. They can be scattered between the two places as above. They will disappear if they are between the 'accountname' specification and a left parenthesis. Note that the 'accountname' specification must preceed all left parentheses, and that only one 'accountname' specification is allowed. Below are legal and illegal examples.

Any alphabetic or numeric option or 'accountname' may be used with any processor, but it will be ignored by the processor if it has no meaning.

OPTION SET	OPTIONS	N	M	acct name	error
>VERB ABC	ABC	DN	DM	no	
>VERB CD (E	CDE	DN	DM	no	
>VERB (EFG	EFG	DN	DM	no	
>VERB G H I	GHI	DN	DM	no	
>VERB I,J,K	IJK	DN	DM	no	
>VERB 3KLM	KLM	3	DM	no	
>VERB 4 M(NO	MNO	4	DM	no	
>VERB (OP5Q	OPQ	5	DM	no	
>VERB Q (R5S	RS	5	DM	no	error
>VERB STU (6		6	DM	no	error
>VERB UVW 7-8	UVW	7	8	no	
>VERB 8-9(WXY	WXY	8	9	no	
>VERB YZ (9-10A	A	9	10	no	error
>VERB (ABC1-2	ABC	1	2	no	
>VERB 1-C2DE	CDE	2	DM	no	error
>VERB 2-E(FG3	FG	3	DM	no	error
>VERB GHI 4 'accountname'	GHI	4	DM	yes	
>VERB 'accountname' (IJK5-6	IJK	5	6	yes	
>VERB K7-8 'accountname' (LM	KLM	7	8	yes	
>VERB 'accountname' M8-9 (NO	NO	DN	DM	yes	error
>VERB 'accountname' (OPQ9-10	OPQ	9	10	yes	
>VERB Q1-2 (RS 'accountname'	QRS	1	2	no	error

Spooler options processor alternatives.

It is syntactically legal to make the second operand smaller than the first, except that the processor will trap to an error because all of the processors operating with a range operate in ascending order.

7.15 CONSIDERATIONS ON PROC CONTROL OF THE SPOOLER.

There are some facilities for the control of the spooler from PROC. Transfer of information concerning the state of the system is into the PROC secondary input buffer.

The spooler system may be controled to some extent from PROC by making use of the information which appears in the secondary input buffer from the error message handler and from certain exceptional information transfers executed by some informational elements of the spooler.

The string of parameters which are left in the secondary input buffer after the execution of a verb, which occurs after the P in the PROC, may be seen by inserting an SS instruction in the PROC after the P, and then a DO. This will display the PROC secondary input buffer. It is of use when debugging PROCs which use the buffer, and during consideration of designs which might use the buffer.

Retreiving the information from the secondary input buffer for use by PROC is related to the E command in PROC, except that the spooler system may occasionally deliver more than one piece of information. The E command will inspect only the first data field in the PROC secondary input buffer. Therefore, the strategy is to point the PROC input pointer at the secondary input buffer by executing an SS command, and then execute a search.

There are three things which may be done. One may search for a number within certain limits of admissability, one may search for a specific number, or one may transfer all or part of the buffer to an output buffer. Note that the only thing which may be done with the secondary input buffer is to inspect it or to transfer it to an output buffer, and that it will disappear at the execution of each PROC verb execution. Therefore, if the information affects more than one verb execution, or if it is to affect a verb execution after the next immediate verb execution, the contents of the buffer must be saved. It may be saved by moving it to the secondary output buffer, which is then fed to a BASIC program, which then inspects, files, or returns it through a DATA or CHAIN statement. If particular pieces of information are expected, conditional statements may be defined which write a field to the primary input buffer upon encountering the specified information.

Information which is passed by means other than the ERRMSG processor are the entry number for each print file control record at print file initialization time, a large volume of data on the condition of the printers from the LISTPTR verb, and tape ownership data from any process which attempts tape attachment. All verbs which use the ERRMSG file leave residues in the form of error message numbers as well. These error message numbers will be left at error message printing time.

7.15.1 CASES OF PROC INTERACTION.

The principle cases which use the interaction are hold file acquisition, tape control, and printer control.

7.15.2 HOLD FILE RECOGNITION.

Processes running under PROC control with an SP-ASSIGNment which specifies that print files are to be held will leave a mark and the print file number in the PROC secondary input buffer at the time when the ENTRY # is being displayed. The mark is the error message number 1099. The print file number is the parameter which succeedes the 1099. If the process generates the message

HOLD ENTRY # 17

then the PROC secondary input buffer will contain at least

1099 17

It will also contain any other error message numbers which may have been generated during the execution of the verb.

What is not contained in the string is the print file identification number as referenced in PICK/BASIC in the PRINT ON statement, in RUNOFF in the .PFILE statement, and with the R parameter in the SP-ASSIGN statement. Control in that case must be due to a well-known order of initialization of print files, either within the verb activation, or by execution of a sequence of SP-ASSIGN Rn's prior to the verb's activation.

An example of a fragment of a PROC to retrieve the hold file entry number for a succeeding SP-EDIT is below.

```
<verb which generates a print file>

P                       Execute the verb.
SS                      Set the buffer pointer to the secondary input
                        buffer.
B
5 IF # A G 99           End of buffer -- entry not found.
IF A = 1099 G 15        Test for mark.
F                       Advance the pointer to the next argument.
G 5                     Test the next parameter.
15 F                    Advance the pointer to the print file number.
IF A # (0N) G 98        Absent or spurious data.
HSP-EDIT                Go SP-EDIT the print file.
A                       Move the print file number to the primary output
                        buffer.
STON                    Turn on the stack for the prompts, as necessary.

P                       Go process the SP-EDIT.

<Process next or exit.>

98 XBAD DATA.           Exit error message.
99 XNO DATA.            Exit with error message.
```

Example of the PROC use of the hold file entry number.

7.15.3 TAPE CONTROL.

The magnetic tape drive unit has always been an exceptional part of the computer system, since, unlike essentially all the other devices and structures in the system, it can not be shared. Therefore, its availability and condition has always been of great importance to any sequence of processes running under PROC control and which use the tape drive, since PROCs can not mount tape, and they can not continue when the tape is inoperative. Note that the tape control error message numbers are 90 through 99, and are to be found in the ERRMSG file. Also note that certain tape difficulties are not ammenable to PROC control because they interrupt the process and speak only to the operator of the terminal. In these cases, only human intervention is capable of dealing with the problem, or human judgement is considered necessary.

Because of the nature of the tape, it is advisable to specifically attach the tape each time it is used, by means of the T-ATT verb. The T-ATT verb will return one of two possible patterns. It the tape is available or is already attached, it will return

90 nnnn

in the PROC secondary input buffer. The number 90 is a mark which specifies that the tape is attached. The string 'nnnn' is the tape block size as it is displayed on the screen. It is in general advisable to use an explicit block size with the T-ATT verb, because there may be exceptional cases when the prior state of the process is unanticipated.

If the tape is not available, the T-ATT verb will return

95 nn

in the PROC secondary input buffer, where 95 is a mark indicating that the tape is not available, and the string 'nn' is the number of the line which has the tape attached.

The second concern with the tape has to do with the modification of the spooler which removes the tape drive from spooler control. If the intent is to send the print file to tape on completion, under PROC control, in a situation wherein it is inconvenient to have the tape drive attached to the print-file generating process, as when several different lines are executing the same function, and all wish to send their current print file to the tape, then it is convenient to send the print file to the tape from the SP-EDIT process at the completion of generation, using the SP-EDIT protocol as noted above, and with at least the T and W options on the SP-EDIT verb. The T option will force the output to tape under a SPOOL Y condition, and the W option will cause the process to wait until the tape drive becomes available.

If a fully automatic process is desired, the the MS options may be added to the SP-EDIT verb. Below is an example.


```
<verb which generates a print file>
P                   Execute the verb.
SS                  Get the print file entry number.
B
5 IF # A XNO DATA.
IF A = 1099 G 15
F
G 5
15 F
IF A # (0N) XBAD DATA.
HSP-EDIT            Go SP-EDIT the print file.
A                   Move the print file number to the primary output
                    buffer.
H MSTW              Spool to Tape; Wait until the tape is available.
PP                  Yeilds SP-EDIT n MSTW

<Process next or exit.>
```

Example of SP-EDIT to tape in PROC.

7.15.4 PRINTER CONTROL UNDER PROC.

Given that this spooler sub-system is both flexible and fairly complex, it is probably preferable in a normal application environment to retain most or all of the spooler manipulation under control of PROCs written by the application system programmer. It is particuarly important since the printer control verbs allow considerable facilities management opportunities.

The principle devices to be used to manage the flow to the printer are the output queues to which the generating tasks are assigned under the SP-ASSIGN verb, and the STOPPTR and STARTPTR verbs which consume the print files according to instalation management plans. It is trivial to control the flow of print files onto output queues. What is more delicate is the reallocation of printers, since a printer must have completed its current task before it can be reallocated using the STARTPTR verb. It is generally unsatisfactory to use the SP-KILL verb indiscriminantly, because necessary reports will tend to be truncated thereby.

The verb which interrogates the condition of the printers and which communicates to PROC is the LISTPTR verb. See the section on the LISTPTR verb for the details of the display and the string which is returned to the PROC secondary input buffer. The LISTPTR verb will return information about the condition of the desired printer, and about the other possible printers on the system. One wishes to know whether the intended printer is allocated; if so, to what forms, whether it is active or inactive, and whether it is stopped. One also wishes to know whether any other printers are allocated to the contemplated output queue number. If one or more is, then the print file may go to one of them instead of the intended printer. In order to discover the condition of the intended printer, say printer 3, then one executes a procedure of the following sort:


```
HSTOPPTR 3              Issue a stop command.
P                       Execute it.
SS                      To the secondary input buffer.
B
5 IF # A G 20           Out of data
IF A = 1171 G 80        Stopped and inactive; go start.
IF A = 1174 G 80        Unallocated; go start.
IF A = 1172 G 15        Go pause
F                       To next buffer element
G 5                     Go test next.
15 HSLEEP 1             Pause
20 HLISTPTR 3           Execute the LISTPTR verb.
P
SS                      Set the pointer to the secondary input buffer.
B
25 IF A = 1034 G 10     Check for the correct initial mark.
IF # A G 99             Not there.
F                       To the next element.
G 25                    Test it.
10 F                    To the condition
IF # A G 99             Nothing there.
IF A = 1074 G 80        Not allocated; can be used.
IF A = 1171 G 80        Inactive; go start.
IF A = 1172 G 40        Active
C The buffer contains illegal data.
XBAD DATA.              Error exit.
40 OTHE PRINTER IS ACTIVE; DO YOU WISH TO WAIT (Y/N)+
SP                      Set the pointer to the primary input buffer.
IP?                     Input the answer.
IF A = Y G 15           Go pause.
X                       Else exit.
80 HSTARTPTR 3,7,0,S9   Start the printer as per the standard.
P                       Execute the verb.
```

Example of the use of the STOPPTR, LISTPTR, and STARTPTR verbs under the control of PROC.

A more complex case is probably best handled using a basic program, which inspects the data and constructs verb strings to which the process then chains.

7.16 MAGNETIC TAPE FACILITIES.

There are three types of magnetic tape verbs: prerequisites, utilities, and production. The production verbs fall into four categories.

The T-ATT and T-DET verbs are prerequsites for the orderly sharing of the magnetic tape drive.

The PICK System provides a complete set of magnetic tape commands which apply to all 1/2 inch reel tape devices. However, not all of the commands will function for all magnetic tape devices. For example, not all Audio Cassette and 3M 1/4 inch cartridge tape devices are capable of T-BCK. In addition, some of these devices are only capable of writing new data at the beginning of the tape (BOT) and the end of data; records in-between cannot be modified.

The T-FWD, T-BCK, T-EOD, T-REW, T-SPACE, T-WEOF, and T-CHK verbs are useful to position, mark and check tape. The T-READ and T-RDLBL verbs are available to inspect tapes, and are the tools available if there are any difficulties encountered using tape as a storage or transport medium.

The first of the production facilities has to do with saving and restoring the system. The storage intitiating verb is SAVE, a discussion of which is to be found in chapter 10 of this manual. With it are associated the ACCOUNT-SAVE PROC, which is the SAVE verb with one selection of options, and the FILE-SAVE PROC with a different selection of options.

Retrieval of data on a tape generated by one of the variations on the SAVE verb is either by 1) the file-restore option of the coldstart tape, 2) the :FILES verb, and which is the same without a coldstart, which has no real point on a orderly system, and 3) the SEL-RESTORE verb, which allows access to items in a specified file. Each of these processors is discussed in chapter 10, infra.

The second of the production facilities is the T-DUMP and T-LOAD verb pair, discussed in this chapter, infra. The T-DUMP verb formats the data on the tape in a very different way than the SAVE verb, so that tapes generated under the SAVE verb may not be read by the T-LOAD verb, and tapes generated by the T-DUMP verb may not be read by verbs intended to work with tapes generated by the SAVE verb. T-DUMP is an ACCESS verb, so that it allows selection of items on the basis of data values and by item lists. It will also work from lists generated by SELECT and the processors related to that verb. The T-DUMP verb generates a tape without the control data included by SAVE, and with the items concatenated end-to-end and spanned. It saves data from one file at a time, and is recommended as the basic data transfer tape writing verb. T-LOAD is the verb which retrieves data written by T-DUMP. It is also capable of selection criteria, which cause only specified records to be written to disk.

The third set of facilities are primarily for print files. All processors which can produce printed output, can send the print file to tape. Any print file which can be SP-EDITed can be sent to tape. This has been discussed above. Retrieval of print files from tape is by means of the

SP-TAPEOUT verb which is discussed below. It is possible to build tapes which contain data readable by non-PICK machines by using a print file generating process; and it is possible to move data from machines which delimit data records on tapes with a carriage-return, line-feed sequence by means of the SP-TAPEOUT verb and the print file to data file facilities of the SP-EDIT verb.

The fourth set of facilities for tape generation and retrieval are the operators in PICK/BASIC which read and write tape. Unlike the other tape processes used by the system, PICK/BASIC has control of all blocking and deblocking of tape data records. Each time that the WRITET operator is envoked, the string specified as the operand is placed in the next tape block, and any unused portion of the tape block is padded with blanks. If the string which is to be sent to the tape is null, an error message will be issued. If the length of the string which is being sent to tape exceeds the size of the tape block, an error message will be issued, the string will be truncated, and as much of the string as will fit in the tape block will be written to tape. Under PICK/BASIC there are no automatic spanned records. It is therefore convenient to use fixed-length records, which may be either blocked or unblocked, and it is efficient to specify the tape record length as the size of the block of data being written by the PICK/BASIC program. If the data is naturally of variable length, then a spanned-record protocol must be constructed in PICK/BASIC and managed by the PICK/BASIC program.

7.16.1 COMMUNICATION WITH OTHER PICK-CLASS MACHINES.

In general, and with no assurances that future machines of this class will continue to behave as follows, the most stable tape transfer facility is the T-DUMP, T-LOAD verb pair, using a tape block size of 500 bytes, and without a tape label. The tape block size of 500 bytes is the minimum size of core buffers in PICK-class machines, and as such, assures backward compatability. Tape labels vary between different versions and releases.

7.16.2 COMMUNICATIONS WITH NON-PICK-CLASS MACHINES.

Dealing with an abitrary foreign machine can be unpleasant. The first thing to note is that the tape drives must be compatable. PICK normally uses 9-track, 800-bpi, NRZI tape drives, with an ASCII character set. 9-track means that all 8 bits of one byte and one non-error-correcting lateral parity bit are placed across the tape at the same time. 800-bpi means a data density of 800 data locations per inch. The bpi means "bits per inch" along one of the 9 tracks on the tape. Another term is "800-cpi", where "cpi" means a more accurate "characters per inch". NRZI means "non-return zero", and is a particular physical magnetic tape recording protocol. The ASCII (American Standard Code for Information Interchange) is the collection of bit patterns used to represent characters in the PICK machine, as opposed EBCDIC or another code pattern. It is typified by noting that all characters are in the lower 128 possibilities, that the numbers collate before the alphabet, and that capital letters collate before lower-case letters. Another tape protocol which occurs on some PICK machines is 1600-bpi PE (Phase Encoded).

The two tape drives involved in the transfer must be compatible. If they are not, then a third machine with tape drives compatible with each must be found, and a second tape must be made which is a translation of the first from one tape protocol to the other. This third machine is called a service bureau.

Once the physical protocols are compatible the character sets must be translated. PICK includes facilities for translation to and from EBCDIC. If the PICK machine is the destination, and the tape is in a print file format, then SP-TAPEOUT may be used with the A option, and the print file to data file facility of SP-EDIT may be used to move the data to a data file. If the data is not in print file format, the PICK/BASIC must be used to read the tape, execute the deblocking, and form the PICK-class items. The operator ASCII can translate the string from EBCDIC to ASCII.

If the PICK machine is the source, then it is usually convenient to build a tape of the form which the destination machine requires. If print file format is acceptable, the various facilities of PICK/BASIC, ACCESS, and RUNOFF may be use to make the tape directly, or through a hold file. If the destination machine requires EBCDIC, it is then convenient to use the hold file to data file protocol of the SP-EDIT verb, and then do the translation using PICK/BASIC. If a print file format is not convenient, then PICK/BASIC must be used.

Usually, PICK/BASIC power and flexability is highest when arbitrary data patterns are required. It may build fixed length records (using the form "R#10", say), to translate, to block, and to deblock.

Tape labels are generally not recommended when transferring data to foreign machines, because their tape label protocols are different, and often intransigent, and becuase tape labels have a different block size than will probably be used for the data (80 bytes). When transferring data to an PICK machine, tape labels are also not recomended. Because they do not fit the PICK protocols, they cannot be read by T-RDLBL; and since they are not considered to be labels, they will be considered to be data. In this case, the reading process will either receive spurious data, in the case that the label tape record size is greater than or equal to the size of the data records on the tape, or a BLOCK TRANSFER INCOMPLETE error message will be issued. All is not lost, however, because the label can usually be read by the T-READ verb, and because it can be skipped by using a T-FWD 1.

In order to use the T-READ verb on the label, the tape block size of the process must be set to match the size of the label. In general, if the tape block size of the process is less than the size of the tape record, then the number of bytes specificied by the T-ATT (tape block size) will be returned to the terminal. If the tape record size is exceeded by the tape block size specified by an execution of the T-ATT verb (or by the default or residual size -- see the discussion of T-ATT infra), then the BLOCK TRANSFER INCOMPLETE message will be issued. It is therefore possible to discover the size of tape records, either label or data. One does a binary search from a T-ATT specification which does not cause an error to a T-ATT specification which does cause an error by splitting the difference each time, and going toward the case which did not occur. It is the case that no error will be reported if the T-ATT specification is one byte larger than the actual tape record size. It is also the case

that if the T-ATT specification is set to a tape block size less than the actual tape record size, then a number of bytes in the block equal to the difference in the to sizes will be lost from the end of the tape record. The reading process may run to completion, but data will have been lost. It therefore behoves one to test the block size of the data before actually reading all of it into a file.

It is also convenient to use the T-READ verb to inspect the data on the tape, to see if it is as advertised. If it is in EBCDIC, it may be translated to ASCII by means of the A option. If it is in binary, it may be displayed in hexidecimal by the use of the X option. It may be sent to the printer by means of the P option. See the discussion of the T-READ verb infra.

Given the ability of the PICK machine to inspect tape, control label generation, and form and translate data, it is usually convenient to do the conversion on the PICK machine, rather than the foreign machine.

7.17 MAGNETIC TAPE: TAPE RECORD SIZE

The T-ATT verb attaches the tape drive to a user's process and specifies the tape record length. This section notes considerations on tape record size selection.

The T-ATT verb, noted in the next section, allows the user to specify the length of the block sent to the tape on each write, or retrieved from the tape on each read. The default value is 4000 bytes. The exceptional default is 500 bytes for the T-DUMP verb for backward coompatability. The range of allowable values is arbitrary, but may be taken to be from 80 bytes to 24768 bytes for the following reasons.

Tapes are written with a fixed physical gap between records known as the inter-record gap (IRG), thereby placing an upper limit on the number of records on a tape of given length. The length of each tape record is a function of the number of bytes per record, however. Therefore increasing the record length increases the storage capacity of the tape. Furthermore, since the tape drive starts and stops for each record, increasing record size decreases the number of records containing a body of data, decreases wear and tear on the tape drive, decreases processing time, and increases system throughput.

Because data files, items, and their elements are of variable length, the length of tape records has no influence on the contents of blocks of data written to tape or read from tape and vice versa. Thus the tape routines automatically handle multiple item records and spanned item records. This compact protocol is used by the various system tape processors: the file save and restore and T-LOAD and T-DUMP. The various logical records and their elements are delimited with the usual system delimeters.

The protocol for the PICK/BASIC READT and WRITET are different. Basic handles the tape by writing one tape record for each logical record. This is useful when conversing with a fixed-length-record machine, and when storing data on tape which is to be returned through a basic program. When conversing with a fixed-record-length machine it is useful to define the tape record size so that it matches the length of the data being sent to it, and it is efficient to block the data being sent. Blocking data normally means the process of placing a well-known number of logical records of fixed and equal length in a single physical block for the convenince of the physical device on which it is to be stored. Retreival of the data then requires deblocking into individual logical records. The PICK tape processor will supply only &One\ Tape record for each record sent by a WRITET instruction. If the data record overflows the tape record, it will be truncated on the right and an error message will be issued. If the data record does not fill the tape records, the remainder of the last tape record will be padded with blanks.

It is possible for the user to define the tape record size for print files. Note that print files are a continuous string of data, with lines delimited by carriage-return (X'0D') and line-feed (X'0A') characters. Pages are delimited by form-feed (X'0C') characters. A line of a print file is stored through the last non-blank character. There is no padding of blanks on the right. The end of a print file is padded with nulls (X'00'). This suggests that communication of a print file to a foreign machine requires the acceptance by the foreign machine of the four

characters noted above, and that it not require fixed-length records. If there is some problem with this set of protocols, then use of the SP-EDIT process to transfer the print file to a data file, followed by the use of a basic program is suggested. ACCESS will require various strategies if headings, footings and control breaks are required.

If the carriage-return, line-feed characters are useable, it then becomes possible to generate blocked, fixed-length data records using ACCESS, noting that if the page length parameter found in the TERM statement is zero, then ACCESS will not paginate; and that if two bytes are reserved for the CRLF, then there are sufficient resources in the F-correlative structure to generate fixed-length data records.

Another strategy is to construct the meaningful part of the data record using a SELECT or SSELECT with an output attribute, which generates a fixed-length data record using an F-correlative. The list is then saved, and then given to a PICK/BASIC program which uses the READNEXT command to obtain the fixed-length data records. It may them pad them out to a desired length, translate them to EBCDIC, and block them as desired. This approach appears to get the job done with less programming time, and less computer time. The COPY-LIST verb is then available if a hard copy of the transmitted data is desired. T-READ is useful in order to obtain a sample of the tape for verification, and as an aid to the recipient thereof. It does not appear that many non-PICK-class machines have a capability similar to T-READ.

Note that the tape file will be padded on the end with X'FB' characters if it comes from a data transfer verb, that it will be padded on the end with X'00' characters if it comes from a print file generating process, and that it will be padded with blanks by PICK/BASIC.

A minimum record size of 80 bytes is recommended only for the generation of a tape of card images for transfer to a machine which needs such a tape. For off-line storage of data to be used by Pick systems, a record length of 4000 bytes is recommended as the best compromise between tape storage efficiency and the tape record size upper limit considerations, which are as follows.

The T-ATT verb attaches the tape drive to a user's process and specifies the tape record length.

The T-ATT verb is used prior to all tape-control verbs in the following sections, prior to generating print file output to the tape under an SP-ASSIGN T, the execution of an SP-EDIT under SP-ASSIGN T or with the T option, tape reads and writes in PICK/BASIC, or tape output using the REFORMAT verb.

The T-ATT verb is used both to assure that the tape is attached to the user's process before it is used, in order to avoid collisions, and to specify the tape record or block size. Note the preceeding discussion of tape record size. FORMAT:

T-ATT {(N)} or T-ATT {N}

The T-ATT command assigns the magnetic tape unit to the terminal issuing the command. Other users are locked out. If the tape unit is attached to another line, the following message is displayed on the terminal:

TAPE ATTACHED TO LINE n

If attachment is successful, the following message is displayed:

TAPE ATTACHED BLOCK SIZE: n

If already attached to the process, the following message will occur:

BLOCK SIZE: n.

All tape manipulation proceses on the system will check for attachment, attach the tape if possible, generate the required message, and terminate if the tape is not available. The implied T-ATT will use the line's current tape record size specification. If there is no specification, then the default will be 500 bytes. Once the line's tape record size is initialized after LOGON, it will persist until LOGOFF unless it is changed by the use of the T-ATT verb. The tape record length may be initialized explicitly, by the use of the T-ATT with a numeric argument, implicitly by the use of the T-ATT without a numeric argument, by using any tape verb which checks for tape attachment, or by executing the T-RDLBL verb when a labeled tape is mounted, in which case the tape record length specified in the tape label will be transferred to the tape record specification for the line.

none	The tape will be attached to the user's line, if possible, and the tape record length will be the line's current tape record size if it has been initialized or it will be the default 4000 bytes.
n	The tape will be attached, as above, and the tape record size will be n bytes, where n is preferably between 80 and 24768 bytes.

T-ATT options.

T-ATT	Attaches the tape to the user's line if possible. The default record length of 4000 bytes is taken.
T-ATT (80)	Attaches the tape to the user's line if possible. Tape record length is 80 bytes.
T-ATT 4000	Attaches the tape to the user's line if possible. Tape record length is 4000 bytes - recommended.

Examples of the T-ATT {(N)} verb.

Note that the numeric option can be used either with or without a left parenthesis. See the section on options, above.

Message	Meaning and response
BLOCK SIZE: n	The tape continues to be attached to your line. Proceed.
TAPE ATTACHED BLOCK SIZE: n	The tape is attached to your line. Proceed.
TAPE ATTACHED TO LINE n	The tape is attached to another line. Check with the user of that line. Detach if reasonable and possible, then reexecute the T-ATT.

T-ATT messages and responses.

Successful	90 N	where N is the tape block size.
Unsuccessful	95 N	where N is the line to which the tape is attached.

T-ATT error message pattern.

7.19 MAGNETIC TAPE: THE T-DET VERB

The T-DET verb detaches the magnetic tape from the user's line, allowing other processes access to it.

FORMAT:

T-DET {U}

The T-DET verb releases tape unit attachment if the user's line is currently attached to the tape unit. Otherwise, the following message is displayed:

[1147] NOT ATTACHED!

Users with SYS2 privileges may use the unconditional form, T-DET U. This will detach the tape unit from any line except the spooler, which detaches automatically at the end of each job. An attempt by a user to use this form without SYS2 privileges, or an attempt to detach the spooler from an error message.

Use of the SP-ASSIGN verb also affects tape attachment or detachment. Successful execution of the SP-ASSIGN T requires that the tape be available, because it is attached at that time. It must then be detached at some later time by the user. The tape will automatically be detached from the user's line when the user logs off the system.

none	Will detach your line if it is attached, else the NOT ATTACHED message will be returned.
U	Will detach any line on the system which has the tape. Requires SYS2 privileges, in lieu of which message 82 will be returned. If the option is allowed, no message will be returned, and the tape will be available whether or not it was in use.

T-DET options.

T-DET	Will detach the tape from your line.
T-DET U	Will detach the tape from any line if your account has SYS2 privileges.

Use of the T-DET verb.

7.20 MAGNETIC TAPE CONTROL: THE T-FWD, T-BCK, T-REW, T-SPACE, AND T-EOD VERBS

This section discusses verbs to be used to space the tape forwards and backwards to the location of the desired file.

T-FWD

FORMAT:

T-FWD {n}

This command moves the tape forward n records. The maximum value for n is 32,767. If n is not specified, the tape spaces forward to the position immediately beyond the next EOF mark.

T-BCK

FORMAT:

T-BCK {n}

This command backspaces the tape n records. If n is not specified, default is a backspace to the position immediately preceding the previous EOF mark, or to the load point; before reading the next record, a T-FWD must then be issued to position the tape after the EOF mark.

Note that both T-FWD and T-BCK use the options handler and that the numeric entry n may be either within parentheses or not as the user prefers.

T-SPACE

FORMAT:

T-SPACE n

This command is a proc which executes the T-RDLBL and T-FWD verbs n times. It spaces the tape forward over n files.

T-EOD

FORMAT:

T-EOD

This command moves the tape forward to the end-of-file mark after the last file in the tape.

T-REW

FORMAT:

T-REW

The T-REW command rewinds the tape unit to the load point (BOT) and returns control immediately to TCL.

T-FWD {n}	spaces the tape for ward n records, n <32,768, or spaces the tape forward to just beyond the next EOF mark, if n is null.
T-BCK {n}	backspaces the tape n records, n<32,768, or back-spaces the tape to just before the last EOF mark, if n is null.
T-REW	rewinds the tape to the load point.
T-SPACE N	spaces the tape forward past n files.
T-EOD	spaces the tape forward past the last file.

Form of the T-FWD, T-BCK, T-REW, T-SPACE and T-EOD verbs.

T-FWD	Spaces forward past next EOF mark.
T-FWD 37	Spaces forward 37 records.
T-BCK	Spaces tape to immediately before prior EOF mark.
T-BCK 22	Backspaces tape 22 records.
T-SPACE 12	Spaces the tape forward 12 files.
T-EOD	Spaces the tape forward past the end of the last file.

Examples of T-FWD, T-BCK, T-REW, T-SPACE, and T-EOD.

7.21 MAGNETIC TAPE CONTROL: THE T-WEOF AND T-CHK VERBS

This section discusses magnetic tape unit commands. Magnetic tape unit control commands are detailed in this topic. Subsequent topics within this section present additional tape commands and features.

T-WEOF

The T-WEOF command writes an end-of-file (EOF) mark on the tape. Its form is:

```
T-WEOF
```

T-CHK

The T-CHK verb is used to check a tape for parity errors. Its form is:

```
T-CHK {(A)}
```

With no option, the verb will cause the file at which the tape is currently located to be checked for parity errors. The processor will return to TCL at the end of the file, and display the message below. The A option will check all files on the tape until it encounters an EOD, which is a double EOF mark and signifies the logical end of the tape. The processor will return to TCL with the message:

```
[91] END TAPE CHECK - n FILE(S)
```

where n is the number of files checked. It will be 1 if no option was used, or the number of files on the tape if the option A was selected.

T-WEOF	Writes an end of file mark on the tape.
T-CHK	Checks a tape-file for parity errors.
T-CHK (A)	Checks a complete tape for parity errors.

Figure A. Form and examples of the T-WEOF and T-CHK verbs.

7.22 MAGNETIC TAPE I/O: THE T-DUMP, S-DUMP AND T-LOAD COMMANDS

The T-DUMP and S-DUMP ACCESS verbs causes selected file items to be dumped (written) to the magnetic tape unit. The T-LOAD command allows the user to load (read) selected items from files previously saved via a T-DUMP or S-DUMP operation.

The T-DUMP and T-LOAD verbs are used to move files and parts of files from one machine to another bu means of magnetic tape. They are also of use if one wishes to back up a particular file rather than a whole account. They are also tape generation and retrevial processors which have been most resistant to change, so that they are of particular use in transporting data between different versions and vintages of PICK class machines. In this case, a T-ATT 500 is recommended. The verbs are also useful if a given large file is to be reallocated without using either an account restore or a system restore, particularly in the case that the disk is almost full, and the source and destination files are at opposite ends of the disk. By dumping the file to tape, and then loading it from tape, the reduction in total disk head travel time will often make the file-to-file copy by way of tape faster in real time and less disrupive to normal system use, than using the copy verb.

The T-DUMP verb copies the specified contents of the specified file from disc to tape. The S-DUMP verb copies the specified contents of the specified file from disc to tape in a sorted sequence. The T-LOAD verb restores the contents of a T-DUMP or S-DUMP tape to a specified file.

T-DUMP and S-DUMP

FORMAT:

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

The S-DUMP verb allows sort criteria as well. The form and operations specified by the "item-list" and "selection-criteria" are described in the ACCESS Reference Manual. They select a sub-set of the items in the specified file which are to be written to tape. Absence of both causes all items in the file to be written to the tape. The HEADER option is used to include the "name" in the tape label written at the start of the file. See the topic entitled GENERATING AND READING TAPE LABELS. The file-name may be preceded by the DICT modifier to dump dictionary data. File definition items (D/CODE=D) will not be dumped. An EOF mark is written to the tape at the completion of T-DUMP. As in other ACCESS statements, each item-id must be enclosed in double quotes. FORMAT:

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

This command allow the user to load dictionaries or data files saved by a T-DUMP operation. The data from the tape are loaded to the file "file-name". The {item-list} and {selection-criteria} options allow the selection of a sub-set of the items in the file on tape for inclusion in the destination file "file-name" on disk. The Dictionary used by the {selection-criteria} processing routines is that of the destination file. Items in the tape file with item-ids identical to items in the destination

File will overwrite only if the (O) option is specified. Item-ids will be listed at the terminal as they are loaded unless the (I) option is used, which suppresses (inhibits) the item-id listing. The tape is positioned at the EOF mark at the conclusion of the operation.

Note that the tape must be positioned at the first record of the file. Otherwise, the tape read will probably commence with an initial spurious item, because the tape records and items are not aligned. See the discussion of tape record length above.

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

S-DUMP {DICT} file-name {item-list} {selection-criteria}
       {HEADER "name"} {sort specifications} {(options)}

T-LOAD {DICT} file-name {item-list} {selection-criteria}
       {HEADER "name"} {(options)}

  DICT                 Specifies dictionary of "file-name" - optional.

  file-name            Name of source file (T-DUMP) or
                       destination file (T-DUMP) - required.

  Item-list            List of items to be dumped or loaded - optional.

  Selection-criteria   Selection of items to be dumped or loaded
                       - optional.

  HEADER "name"        Causes the tape label "name" to be incuded in
                       in the label written - optional.

  Options              Specifies options to be taken - optional.

    (I)                Suppresses (inhibits) listing to the terminal
                       of items dumped or loaded - default is listing.

    (O)                Enables overwrite of items in file with item-ids
                       corresponding to the item-ids of items in the
                       tape file.  Default is retention of destination
                       file items.
```

Form of the T-DUMP and T-LOAD verbs.

T-DUMP FILE1

    <Copies FILE1 to tape. Writes an EOF on completion.

T-LOAD FILE2

    Loads the contents of the tape from the location of the tape at initiation of the T-LOAD through EOF into FILE2. Items on the tape with item-ids identical to those of items in the file are not written into the file.

T-DUMP FILE3 "ltem7" "item11" "item17" HEADER "PRIME" (I)

    Writes the items item7, item11, and item17 in FILE3 (if they exist) to tape after inserting the text "PRIME" into the label, and writes an EOF at completion. The item-ids of the items dumped are not listed on the terminal.

T-LOAD FILE4 WITH VALUE = "37" (O)

    Copies all items in the tape file which have contents such that the attribute "VALUE" in the dictionary of FILE4 evaluates to 37. Any items in FILE4 which have item-ids identical to the item-ids of selected items in the tape file are overwritten.

S-DUMP FILE5

    This command will sort the file into ascending sequence by item-ids, and then transfer it to tape, listing the item-ids on the terminal and write an EOF at the end of the file.

Examples of the T-DUMP and T-LOAD verbs.

7.23 THE T-READ COMMAND.

The T-READ command dumps the contents of the tape to the terminal or line printer.

FORMAT:

T-READ {(options)}

This command dumps the content of the tape to the terminal (or optionally to the line printer). The T-READ operation terminates when the specified number of items (records) have been dumped, or when an EOF mark is detected. Valid options are as follows:

A	-	Dump alphanumeric segment in character after conversion from EBCDIC to ASCII.
X	-	Dump in hexadecimal instead of character format.
P	-	Dump to the line printer.
n{-m}	-	Dump nth through mth tape record, counting from the current position of the tape. If m is omitted, m=n is assumed. If the entire n-m option is omitted, all tape records up to the EOF (or EOT) will be dumped.

The T-READ operation must be preceded by tape attachment to the user's line by to T-ATT verb. The T-READ verb is used to investigate the contents of a tape. It may be used to find out either the contents of the tape in general, or to find the location of a specific file. Note in the examples of the T-READ in the following section that the tape label is displayed at the start of the output. This and the contents of the output generated by T-READ may be used to find the location of a desired file.

In the next section there are samples of the T-READ operation on a fragment of basic program to display the format of the default character form (Figure B) and the optional Hexadecimal form (Figure C). The structure of the label will be discussed in the section on T-RDLBL. It is in the upper left hand corner of each form. Each record is preceded by a record counter. Note that the end of the last tape record is filled with X'FB' ([) after the end of the valid data.

```
T-READ {(options)}   dump tape records to printer or terminal.
                     all tape records in file to terminal in
                     character form.

                     options:
                       A - EBCDIC to ASCII conversion.
                       X - hexidecimal output
                       P - output to line printer.
                       n{-m} - output n recor ds, or
                               record n through record m.
```

General form of T-READ Command

7.24 EXAMPLES OF THE T-READ COMMAND.

EXAMPLES OF THE T-READ COMMAND.

The T-READ command dumps the contents of the tape to the terminal or line printer either in character form or in hexidecimal and character form.

```
L 01F4 16:12:27  20 MAR 1978 BP

RECORD = 1

   1 FORMATC*****************************************
  51 ************************  THIS PROGRAM FORMATS A D
 101 ATA/BASIC PROGRAM TO[*  DISPLAY BLOCK STRUCTURIN
 151 G BY INDENTING LINES.*****************************
 201 ************************************----[ DEFIN
 251 ITIONS10    LOOP          SP = 6           ;* LEFT
 301  MARGIN COLUMN NUMBER          ID = 3           ;*
 351 NUMBER OF SPAC[ES TO INDENT          FLF=0
 401     ;* FUNKY LINE FLAG*---- INITIALIZATION
 451    SPX = SP          LINE.NO = 0*--[-- INPUT FIL

RECORD = 2

   1 E NAME AND  PROGRAM NAME          PRINT          P
  51 RINT          PRINT 'DATA/BASIC FILE NAME    - ';
 101 INPUT FI[LE      UNTIL FILE='' DO          OPEN
     .
     .
     .
 401 [[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[
 451 [[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[

[94]  END OF FILE
```

FIGURE B. T-READ in character form.


```
L 01F4 16:12:27  20 MAR 1978 BP

RECORD = 1

0000  464F524D 415443FE 2A2A2A2A 2A2A2A2A      0 :FORMATC^********:
0010  2A2A2A2A 2A2A2A2A 2A2A2A2A 2A2A2A2A     16 :****************:
0020  2A2A2A2A 2A2A2A2A 2A2A2A2A 2A2A2A2A     32 :****************:
0030  2A2A2A2A 2A2A2A2A 2A2A2A2A 2A2A2A2A     48 :****************:
0040  2A2A2A2A 2A2A2A2A FE2A2020 54484953     64 :********^*  THIS:
0050  2050524F 4752414D 20464F52 4D415453     80 : PROGRAM FORMATS:
0060  20412044 4154412F 42415349 43205052     96 : A DATA/BASIC PR:
0070  4F475241 4D20544F FFFBFE2A 20204449    112 :OGRAM TO_[^*  DI:
0080  53504C41 5920424C 4F434B20 53545255    128 :SPLAY BLOCK STRU:
0090  43545552 494E4720 42592049 4E44454E    144 :CTURING BY INDEN:
00A0  54494E47 204C494E 45532EFE 2A2A2A2A    160 :TING LINES.^****:
00B0  2A2A2A2A 2A2A2A2A 2A2A2A2A 2A2A2A2A    176 :****************:
00C0  2A2A2A2A 2A2A2A2A 2A2A2A2A 2A2A2A2A    192 :****************:
00D0  2A2A2A2A 2A2A2A2A 2A2A2A2A 2A2A2A2A    208 :****************:
00E0  2A2A2A2A 2A2A2A2A 2A2A2A2A FE2A2D2D    224 :************^*--:
00F0  2D2DFFFB 20444546 494E4954 494F4E53    240 :--_[ DEFINITIONS:
0100  FE313020 2020204C 4F4F50FE 20202020    256 :^10    LOOP^     :
0110  20202020 20535020 3D203620 20202020    272 :      SP = 6     :
0120  20202020 203B2A20 4C454654 204D4152    288 :      ;* LEFT MAR:
0130  47494E20 434F4C55 4D4E204E 554D4245    304 :GIN COLUMN NUMBE:
0140  52FE2020 20202020 20202049 44203D20    320 :R^           ID = :
0150  33202020 20202020 2020203B 2A204E55    336 :3            ;* NU:
0160  4D424552 204F4620 53504143 FFFB4553    352 :MBER OF SPAC_[ES:
0170  20544F20 494E4445 4E54FE20 20202020    368 : TO INDENT^      :
0180  20202020 464C463D 30202020 20202020    384 :    FLF=0        :
0190  20202020 3B2A2046 554E4B59 204C494E    400 :    ;* FUNKY LIN:
01A0  4520464C 4147FE2A 2D2D2D2D 20494E49    416 :E FLAG^*---- INI:
01B0  5449414C 495A4154 494F4EFE 20202020    432 :TIALIZATION^     :
01C0  20202020 20535058 203D2053 50FE2020    448 :      SPX = SP^  :
01D0  20202020 2020204C 494E452E 4E4F203D    464 :        LINE.NO =:
01E0  2030FE2A 2D2DFFFB 2D2D2049 4E505554    480 : 0^*--_[-- INPUT:
01F0  2046494C                               496 : FIL:

RECORD = 2

0000  45204E41 4D452041 4E442020 50524F47      0 :E NAME AND  PROG:
0010  52414D20 4E414D45 FE202020 20202020     16 :RAM NAME^        :
      .
      .
      .
01E0  FBFBFBFB FBFBFBFB FBFBFBFB FBFBFBFB    480 :[[[[[[[[[[[[[[[[:
01F0  FBFBFBFB                               496 :[[[[:

[94]  END OF FILE
```

Figure C. T-READ in hexidecimal form.

7.25 THE SP-TAPEOUT VERB.

The SP-TAPEOUT verb moves print files on magnetic tape to print files on disc. They are handled as though they were generated by any other system processor.

FORMAT:

SP-TAPEOUT {options}

The SP-TAPEOUT verb executes T-ATT and inputs the contents of a tape file to the Spooler. Disposition of the file input to the Spooler is according to the current SP-ASSIGNment of the user's line. It may be printed immediately, at the completion of input, or as choked input. It may be saved as a hold file. It may be then be returned to tape from the hold file through the facilities of the SP-EDIT verb.

The SP-TAPEOUT verb has two options. The U options causes all alphabetic information to be masked to upper case. The A option causes an EBCDIC to ASCII to occur. The A option is executed before the U option; and both can be executed on the same tape file. All SP-TAPEOUT manipulations will assume process print file 0. Each print file on tape is normally moved to the disc as a distinct print file; if they are to be placed into one disk print file, then the SP-ASSIGN 0 option should be used. In general, if it is desired to concatenate several print files in to one print file, the only way currently available to do it is to send them to the tape by way of SP-EDIT, and then to return them under SP-ASSIGN 0 using SP-TAPEOUT.

SP-TAPEOUT will fail under SP-ASSIGN T. Note that the SP-ASSIGN CI process is available to limit disc usage by the print file.

A	Causes conversion from EBCDIC to ASCII between tape and the print file. A
L	Causes print files which have been transmitted with one line per tape record, right padded with blanks, and without chariage-return, line-feed sequences embedded in them to be transferred to spooler print files directly, if the tape record length is less than or equal to 140 bytes. The option causes each tape record to be treated as a line. Trailing right blanks are removed, and a carriage-return, line-feed sequence is inserted.
U	Causes conversion of all lower case alphabetic characters to upper case
AU	Causes conversion from EBCDIC to ASCII and then to upper case. AU

SP-TAPEOUT options.

SP-TAPEOUT	Spools the file at the current location of the tape to the destination specified by the user's current output assignment.
SP-TAPEOUT U	Spools the print file on tape to the specified destination, converting lower case to upper case in the process.
SP-TAPEOUT A	Converts the print file on tape from EBCDIC to ASCII and spools the converted print file to the specified destination.
SP-TAPEOUT UA	Converts the print file on tape from EBCDIC to ASCII, then converts the ASCII to upper case, and then spools the print file to the specified destination.
SP-TAPEOUT L	Causes each tape record to be treated as a line. Trailing blanks are removed, and a carriage-return, line-feed sequence is inserted.

Examples of SP-TAPEOUT.

7.26 THE T-RDLBL COMMAND. GENERATING AND READING TAPE LABELS

Tape labels may be generated via File-Save, S-DUMP, or T-DUMP operations. Tape labels may be read by File-Restore, T-LOAD, or T-RDLBL operations.

GENERATING TAPE LABELS

A tape label may be written at the beginning of each tape reel. The system adds the time, date and the reel number. Labels are specified when invoking either the File-Save, S-DUMP or T-DUMP processor. In the case of a File-Save, the label is written after the Cold-Start section of the tape. T-DUMP and S-DUMP will cause the label to be written only if the tape is at the load point. It will be ignored otherwise.

Labeled tapes are always created via the T-DUMP or S-DUMP ACCESS verb; by using the HEADER option the user may specify text to be included in the label. For example,

```
T-DUMP FNA HEADER "XYZ"
```

causes the text "XYZ" to be included in the tape label. Note that the label must be enclosed in double quotes.

READING TAPE LABELS

The tape label is read by a File-Restore, a T-LOAD, or a T-RDLBL operation. The File-Restore operation will always read the first record presented to it when an "A", "AF" or "F" option is entered to see if it is a label. The T-LOAD command (see below) will attempt to read the label only if the tape is at the load point. In the case of unlabeled tapes, the operations will read the first tape record, determine that the tape is unlabeled, and backspace the tape by one record before continuing.

THE T-RDLBL COMMAND

FORMAT:

```
T-RDLBL {(N)}
```

This command will read and store the label from tape reel number n (n is hexadecimal), if the tape is at load point. This command must be used to initialize the internal label storage area, and is needed under either of the following conditions,

- If data is to be read from a tape (e.g., T-LOAD, SEL-RESTORE, etc.) starting at other than the load point of reel number one.

The tape manipulation processes normally obtain the tape label at the beginning of each tape and after each end-of-file mark. If a tape manipulation verb is initiated on a reel of a multi-volume set of tape which is not the first, the tape process will obtain the tape label and with it the reel number. At the end of the tape, the process will prompt for the next reel, based on the reel-number of the preceeding tape.

THE FORM OF TAPE LABELS

A tape label consists of eight elements, as shown in Figure B. They are a uniform 80 bytes in length. The label section is 49 bytes long and is constructed by concatenating the source file name in the T-LOAD verb and the "name" in the HEADER "name" option of the T-LOAD verb with an intervening space, and then truncating the right end of this string as necessary to insert it into the label block. If the spooler generates the tape output, the file name is "SPOOLER".

T-RDLBL {(N)}	!reads and stores label from tape reel ! !number (N). (N) is hexadecimal. !

General form of T-RDLBL Command

L 01F4 16:12:27 20 MAR 1978 FILENAME HEADERNAME ... ^01

Element	Source and use
L	Label specifier.
01F4	Record length in hexadecimal. Here, 500 bytes.
16:12:27	System time at tape write.
20 MAR 1978	System date at tape write.
FILENAME	Name of the source file, either filename if T-DUMP was used or SPOOLER if the spooler was used.
HEADERNAME	The "name" in the HEADER option of T-DUMP.
^01	The reel number in bytes 79 and 80.

Contents of the tape label.

SECTION 8

ICON/PICK RUNOFF

ICON®

Chapter 8

RUNOFF

THE PICK SYSTEM

USER MANUAL

8.1 RUNOFF INTRODUCTION AND RUNOFF VERB FORMAT

RUNOFF is a verb which facilitates the preparation and maintenance of textual material such as memos, manuals, etc. RUNOFF takes text prepared with the PICK EDITOR and produces formatted output. RUNOFF source text contains commands which control justification, page titling and numbering, spacing and capitalization. Textual material prepared with RUNOFF may be easily edited and corrected with the PICK Editor and then reprinted with RUNOFF. Material may be inserted or deleted, while unchanged text need not be retyped. RUNOFF also provides the capability of combining separate textual material into a single report and inserting duplicate text into different reports.

RUNOFF is the TCL-II verb issued to process one or more source text file items in RUNOFF format. Multiple input items are treated as a single source text file. A source text item may contain a command which causes RUNOFF to CHAIN to another file item. This makes it possible to CHAIN file items together without doing a SELECT or SSELECT. items included in the RUNOFF verb's item-list may chain to other items within the same file. when the chain ends, processing continues with the next item from the item-list.

A source text item may also contain a command which causes RUNOFF to READ a second file item and then resume processing of the first item. This makes it possible to insert the text from a single file item in the output from many other file items (see example below).

The RUNOFF verb format is: RUNOFF file-name item-list {(options)}

OPTIONS:

C The C option suppresses the .CHAIN & .READ commands.

I The I option will output the name of the next item to be 'Runoff'. (helpful for tracing .CHAINed sequences)

J The J option will suppress Highlighting.

N The N option will cause output to the terminal to be continuous; that is, RUNOFF will not pause at the bottom of a page and wait for a carriage-return if the N option is used.

Nnn This numeric option may be used to set the number of times BOLDFACE letters are overprinted.

P The P option may be used to direct output to the line printer.

S The S option may be used to suppress underlining and boldface when RUNOFF output is directed to a CRT.

U The U option will force the output to upper-case.

The source file contains the textual material which will appear on the final copy, plus command information to specify formatting and alternate sources of input.

Each line of input source text is processed in the text mode except those beginning with a period. A line beginning with a period is assumed to be a command line and is processed in the command mode. A command line may contain one or more commands, each starting with a period. The commands provide formatting information and select various modes of operation.

RUNOFF fills each output line by adding successive words from the source text until one more word will not fit on the line. The line is then justified by inserting blank spaces between words at random until the last word in the line exactly meets the right margin.

RUNOFF may be set to fill output lines without justifying the right margin. When filling lines, spaces and end-of-lines are treated only as word separators. Multiple word separators are stripped from the input.

RUNOFF may be set to transmit the input source text to the output without filling lines or justifying margins. In this mode, multiple spaces and end-of-lines are not stripped from the input. Some of the commands cause a BREAK in the output. A BREAK means that the current line is output without justification. This occurs at the end of paragraphs.

```
.SK.BOX 1,78.SK

.BP.J:PARAGRAPH 0.LINE LENGTH 74.LEFT MARGIN 2

.SECTION 2 INTRODUCTION TO RUNOFF

.INDEX 'RUNOFF Introduction'

.BOX OFF.C
```

Common RUNOFF Commands.

RUNOFF commands are stored along with the textual material in the source file. These commands are distinguished by a period at the start of a command line. A command line may contain one or more command, each starting with a period. The commands provide formatting information and select various modes of operation.

Note: In the following descriptions of RUNOFF commands, valid command abbreviations are enclosed in parantheses (where such abbreviated forms of the command exist).

8.3.1 BEGIN PAGE (BP) BEGIN PAGE (BP)

BEGIN PAGE causes a BREAK (see below) followed by an advance to a new page. The page number is incremented and the page heading (if set) is printed.

8.3.2 BOX n,m / BOX OFF (BOX) BOX n,m / BOX OFF (BOX)

The BOX command causes the following text to be enclosed in a box with the width parameter specified by 'n' (right margin) and 'm' (left margin). The text will continue to be 'boxed' until a "BOX OFF" command is encountered.

For example:

```
001 .BOX 4,74.CENTER
002 This is an example of a BOX.
003 .BOX
```

```
|                    This is an example of a BOX.                    |
```

8.3.3 BREAK (B) BREAK (B)

BREAK causes any partially filled line to be output before processing the next input line.

8.3.4 CAPITALIZE SENTENCES (CS) CAPITALIZE SENTENCES (CS)

This command puts RUNOFF in the capitalize sentences mode. In this mode the first letter of each sentence is capitalized. The first letter after a '.', '?', or '!', followed immediately by either a space or an end-of-line (Attribute Mark) is capitalized. The capitalize sentences mode also causes the following characters to be followed by a double space or an end-of-line: '.', '?', '!', ':', and ';'. CAPITALIZE SENTENCES is one of the STANDARD settings. (See the STANDARD command.) note: This command is ignored in the NOFILL (NF) mode.

8.3.5 CENTER (C)

CENTER causes the next line to be input in NOFILL mode and centered on the next line of output. This command causes a BREAK to occur.

8.3.6 CHAIN {[DICT] [FILE-NAME]} ITEM-ID

This command causes RUNOFF to CHAIN to the input text file item indicated. The [DICT] and [FILE-NAME] are both optional. If DICT is not specified, the DATA section of the file is assummed. If no FILE-NAME is given, the item will be read from the same file as the item being processed.

The text input from this item is processed and output without any parameter or mode changes. RUNOFF does not resume processing text from the current source of input. This command does not cause a BREAK.

The .CHAIN command will scan the string following the command, looking for an item-id or a file name. The legal delimiter for the item-id or file name is a blank. They may have an included period. If there is more than one string following the CHAIN command which is delimited by a blank, then the next-to-the-last field will be taken to the the file name, and that file will be opened. The last field delimited with a blank will be considered the item-id, and it will be retrieved by the RUNOFF processor to be executed next. You can include a comment statement after the CHAIN, however. Therefore, for the purposes of the CHAIN commands, the line is considered exhausted when the processor encounters an end-of-line mark, or when it encounters a period preceeded by a space.

If the processor opens a file when executing a CHAIN statement, that file will be the file from which all succeeding items are retrieved, until the file is respecified by another CHAIN statement.

The C option will suppress the .CHAIN command if it is desired to RUNOFF one element of a chained or treed structure. The I option will cause the name of the next item to be output by RUNOFF to be placed in the last line of the last item RUNOFF. This is of use with relatively large documents.

8.3.7 CHAPTER text

This command may be used to handle automatic chapter numbering and formatting. This command has the same effect as:

```
.BEGIN PAGE.CENTER
.CHAPTER n
.SPACE 2
text
.SPACE 2
```

where the chapter number n is incremented automatically. For example:

```
.CHAPTER RUNOFF
```

would produce:

CHAPTER 8

RUNOFF

8.3.8 ' .* ' THE COMMENT INSTRUCTION

This command informs the RUNOFF processor that all of the rest of the text in the line in which it occurs is a comment. It must either be at the beginning of the line, or after another command in a command line. It is always the last command in a line. This allows text to be commented out, and the intent of READs and CHAINs to be noted.

8.3.9 CONTENTS

This command prints the table of contents accumulated by preceding CHAPTER and SECTION commands. This command should be used at the end of the RUNOFF source file. an example of the results of this command can be seen by looking at the TABLE OF CONTENTS at the beginning of this manual. Note: the LINE LENGTH and LEFT MARGIN of the Table of Contents is determined by those settings that are in effect when the first .CHAPTER or .SECTION command is encountered.

8.3.10 CRT

This command directs the RUNOFF output to the user's terminal. CRT is one of the STANDARD settings. (See the STANDARD command.)

8.3.11 FILL (F)

FILL puts RUNOFF into the line fill mode. Words are processed until there are enough to fill a line without overflowing it. If justifaction mode is on, RUNOFF will insert spaces in the line at random to make the right margin line up. FILL is one of the STANDARD settings. (See the STANDARD command.)

8.3.12 FOOTING

FOOTING causes the next line to be input in nofill mode and stored in a page footing buffer. The page footing buffer will be output at the bottom of each page. The page footing may be changed with successive FOOTING commands. The following characters have special meaning in page footings and headings:

'P' Prints out the page number, right justified in a field of four spaces, with blank fill.

'Pn' Prints out the page number, left justified in a field of 'n' spaces ('n' specified by the user).

'L' Performs a carriage return/line-feed (CR/LF).

'i' Prints out the Item-Id.

'in' Prints out the Item-Id, left justified in a field of 'n' spaces ('n' specified by the user).

'F' Prints out the File-Name.

'Fn' Prints out the File-Name, left justified in a field of 'n' spaces.

'T' Prints out the Time and Date (22 characters long).

'D' Prints out the Date in '01 JAN 1977' format (11 characters).

'C' Centers the line.

FOOTING causes a BREAK and also is one of the STANDARD settings. (See the STANDARD command.)

8.3.13 HEADING

HEADING causes the next line to be input in NOFILL mode and stored in a page heading buffer. the page heading buffer will be output at the top of each page.

The page heading may be changed with successive HEADING commands. the special characters described under the FOOTING command may also be used in page headings.

The HEADING command causes a BREAK and also is one of the STANDARD settings. (See the STANDARD command.)

8.3.14 HILITE c / HILITE OFF

HILITE causes the character specified by 'c' to be printed out at the extreme right margin for every line of text until a HILITE OFF command is encountered. An example of the HILITE command may be seen at the right of this text. *

The highlight command does not cause a break in the text. This allows parts of paragraphs to be highlighted in justify or fill mode. If you wish to align the HILITE command with a paragraph, it may be necessary to put the HILITE | command after the first line of filled or justified text, and to put the form .BREAK at the end of the paragraph. *

The execution of the hilite command also is such that if the term is the last character string in command line, then it is equivalent to HILITE OFF.

The J option will suppress highlighting.

8.3.15 ' - ' TREATMENT OF HYPHENS

Hyphens which are surrounded by alphabetic characters will allow a word-break on the hyphen in fill and justify modes. That is, if a term is a concatenation of two words separated by a hyphen, and the line overflows within the second part of the term, then the first part and the hyphen are left in the line, and the next line is commenced with the second part of the word.

Similarly, if a line in the source text terminates with a hyphen preceeded by an alphabetic character, and the first character in the next line is an alphabetic character, then the last word in the line and the hyphen will be concatenated with the first word in the next line and output together in a line with the hyphen between the two parts. If there is a line overflow which occurs during this process, the hyphenated word will be handled as above. What the processor will not do is remove the hyphen.

If the hyphen does not have this meaning, then the back-arrow character may be placed in front of it to suppress this action.

8.3.16 INDENT n (I)

INDENT causes the next line of output to be indented by n spaces to the right of the left margin. n may be negative to cause the line to begin left of the left margin. If n is missing, n=1 is assumed. This command causes a BREAK to occur.

8.3.17 INDENT MARGIN n (IM)

This command causes the left margin to be increased by n spaces and the line length to be decreased by n. Negative n may be used to decrease the left margin and increase the line length. This command causes a BREAK to occur.

8.3.18 INDEX text

INDEX causes the text specified to be stored in an index list. The text may be more than one word, or several words enclosed in single quotes. The word, or words, along with the current page number, are put in a sorted index list. The index can be printed by the PRINT INDEX command.

8.3.19 INPUT

The INPUT command caused RUNOFF to read the next line of source text from the user's terminal. The text input from the terminal is processed and output without a BREAK or mode change.

8.3.20 JUSTIFY (J)

JUSTIFY puts RUNOFF in the FILL and JUSTIFY mode. RUNOFF fills each output line by adding successive words from the source text until one more word will not fit on the line. the line is then justified by inserting blank spaces between words at random until the last word in the line exactly meets the right margin. JUSTIFY is one of the STANDARD settings. (See the STANDARD command.)

8.3.21 LEFT MARGIN n

This command sets the left margin to n spaces. If n plus the current line length exceeds the maximum line length, this command is ignored. A LEFT MARGIN of 0 is one of the STANDARD settings. (See the STANDARD command.)

8.3.22 LINE LENGTH n

This command sets the line length to n characters (not counting the left margin). If n plus the current left margin exceeds the maximum line length, this command is ignored. A LINE LENGTH of 70 is one of the STANDARD settings. (See the STANDARD command.)

8.3.23 LOWER CASE (LC)

This command puts RUNOFF into lower case mode. in lower case mode all letters are automatically made lower case. They may then be changed to upper case by various text commands or control characters. (See the section on RUNOFF Special Characters.)

8.3.24 LPTR

This command directs the RUNOFF output to the line printer.

8.3.25 NOCAPITALIZE SENTENCES (NCS)

This command resets the CAPITALIZE SENTENCES mode.

8.3.26 NOFILL (NF)

This command resets both the JUSTIFY and FILL modes. Input text lines will be output as they are, (after posssible elimination of special control characters) without removal of extra spaces. Output lines will not be filled nor will right margins be justified. This command causes a BREAK.

8.3.27 NOJUSTIFY (NJ)

This command resets the JUSTIFY mode, but has no effect on the FILL mode.

8.3.28 NOPAGING (N)

The N option may be used to eliminate the wait for terminal input at the end of each page printed on the terminal.

8.3.29 NOPARAGRAPH

This command resets the paragraph mode. blank input text lines and spaces at the beginning of a line will be ignored in justify mode.

8.3.30 PAGE NUMBER n

This command sets the current page number to n. If n is missing, n=1 is assumed.

8.3.31 PAPER LENGTH n

This command sets the paper length to n lines.

8.3.32 PARAGRAPH n

This command causes any blank line or any line which starts with a space to be considered as the start of a new paragraph. This allows normally typed text to be justified without any special commands. n sets the number of spaces paragraphs are to be indented or unindented. A paragraph causes a BREAK followed by (line spacing + 1)/2 blank lines. A PARAGRAPH 5 is one of the STANDARD settings. (See the STANDARD command.)

The PARAGRAPH command may be set to a negative number. Figure A shows the use of a negative paragraph setting to decrease the left margin.

```
001 .SK.PARAGRAPH -4 .LEFT MARGIN 13 .LINE LENGTH 63
002  1.  The user enters the command "Z" to the DEBUGGER prompt character
003 "*". The DEBUGGER responds with "PROG NAME?", the user enters the
004 program name. This allows the DEBUGGER access to the symbol table
005 created during compilation. Alternatively, if the user uses the
006 "(D)" during run time, access to the symbol table is already
007 established, and use of the "Z" command is unnecessary.
008  2.  To find out how far in the loop the program progressed, the
009 user looks at the variable "I" by entering "/I". The DEBUGGER
010 responds with
011 "11 =", at which the user may change the value of "I" if desired.
012 The user may then want to look at all of the values in the array by
013 entering "/ARRAY". The DEBUGGER responds with "ARRAY(1)=1=", the
014 user depresses
015 return and the DEBUGGER continues with the next "array slot"
016 (i.e., "ARRAY(2 etc.)=2=" ). Once "ARRAY(10)=10=" has been reached
017 the . . . etc.
018 .PARAGRAPH 0 .LEFT MARGIN 2 .LINE LENGTH 74
```

Note that in the above example the text lines beginning with 1. and 2. are spaced over one space thus resulting in the negative paragraphing.

The above source text would print:

1. The user enters the command "Z" to the DEBUGGER prompt character "*". The DEBUGGER responds with "PROG NAME?", the user enters the program name. This allows the DEBUGGER access to the symbol table created during compilation. Alternatively, if the user uses the debug option "(D)" during run time, access to the symbol table is already established, and use of the "Z" command is unnecessary.

2. To find out how far in the loop the program progressed, the user looks at the variable "I" by entering "/I". The DEBUGGER responds with "11 =", at which the user may change the value of "I" if desired. The user may then want to look at all of the values in the array by entering "/ARRAY". The DEBUGGER responds with "ARRAY(1)=1=", the user depresses return and the DEBUGGER continues with the next "array slot" (i.e., "ARRAY(2)=2=" etc.). Once "ARRAY(10)=10=" has been reached the ... etc.

Sample usage of a negative PARAGRAPH command.

8.3.33 PRINT INDEX

This command causes the sorted index list of words and page numbers to be printed. The index is sorted into alphabetical order, and printed in two columns per page. Note -- this command changes the tab settings, and causes a BEGIN PAGE command to be performed.

8.3.34 PRINT

The PRINT command causes RUNOFF to print the next line of input text on the user's terminal.

8.3.35 READ {[DICT] [FILE-NAME]} ITEM-ID

This command causes RUNOFF to read the file item indicated. The [DICT] and [FILE-NAME] are both optional. If DICT is not specified, DATA section of the file will be used. If no FILE-NAME is given, the item will be read from the same file as the item being processed. The text input from this item is processed and output without any parameter or mode changes. After processing this item, RUNOFF resumes input with the next line of the current source of input. This command does not cause a BREAK.

The .READ command will scan the string following the command, looking for an item-id or a file name. The legal delimiter for the item-id or file name is a blank. They may have an included period. If there is more than one string following the READ command which is delimited by a blank, then the next-to-the-last field will be taken to the the file name, and that file will be opened. The last field delimited with a blank will be considered the item-id, and it will be retrieved by the RUNOFF processor to be executed next. If the statement is a READ, then the processor will eventually return to this item and continue processing it. When it does, it will commence at the beginning of the next line in the item. Therefore, no statements which occur after the READ statement in the line will be executed. You can include a comment statement after the READ however. Therefore, for the purposes of the READ command, the line is considered exhausted when the processor encounters an end-of-line mark, or when it encounters a period preceeded by a space.

The C option will suppress the .READ command if it is desired to RUNOFF one element of a chained or treed structure. The I option will cause the name of the next item to be output by RUNOFF to be placed in the last line of the last item RUNOFF. This is most useful with large documents.

8.3.36 READNEXT

This command is used to read data from a pre-selected LIST. It has an effect only if, prior to entering RUNOFF, a SELECT, SSELECT, QSELECT or GET-LIST statement has been entered, which selects a list of values. Each READNEXT command in RUNOFF will extract one value from the select-list and place it in the text stream. READNEXT does not cause a break. If there is no pre-selected list, or when the list is exhausted, the READNEXT command will cause a termination of RUNOFF, and a return to TCL.

This command is particularly useful when form-letters are to be generated. For example, it may be necessary to insert the name and address of each recipient of the letter from a separate file. A SSELECT statement is used to extract the relevant data from the file and save it in a list. A series of READNEXT statements will insert the data into the text of the letter. At the end of the letter, a CHAIN statement may be used to restart the next letter. When the list is exhausted, the RUNOFF will stop.

The commands necessary to generate a form letter are:

```
{S}SELECT file-name {selection criteria} attribute-list

.READNEXT

.CHAIN item-name
```

The selected attribute-list contains all the variable information to be 'written' into the form letter. The use of '.READNEXT' commands reads each of these variables and causes them to be 'written' into the letter. The '.CHAIN' command causes the letter to be repeated so long as there is variable information in the selected attribute list. The following example demonstrates the generation of a form letter.

Assume the dictionary of the accounts payable file for a company contains the following three Attribute defining Items:

```
NAME                    ACCOUNT              AMOUNT
001 A                   001 A                001 A
002 1                   002 2                003 3
003 CUSTOMER NAME       003 ACCOUNT TYPE     003 AMOUNT DUE
004                     004                  004
005                     005                  005
006                     006                  006
007                     007                  007
008 A1:","              008                  008 A;3(MR2$,):"."
009 L                   009 L                009 R
010 25                  010 30               010 10
```

The dictionary also contains the following form letter written in RUNOFF:

```
 LETTER

001 .SK 8
002 Dear Mr.
003 .READNEXT
004  Our records show that your
005 .READNEXT
006 account is overdrawn by the amount of
007 .READNEXT
008 We would appreciate prompt payment.
009  Thank you,
010            Indiana Jones
011 .SK 2
012    President CELEBRITY SERVICES CO.
013 .SK 3
014 .BP
015 .CHAIN LETTER
```

The data file contains items such as the following three:

```
 250                    251                  252

 001 Magic Johnson      001 Eddie Van Halen  001 Boy George
 002 Basketball Shoes   002 Guitar String    002 Voice Lesson
 003 25000              003 12345            003 452359
```

To generate the form letter the data file is first sort selected by the name with the attribute list of NAME ACCOUNT and AMOUNT:

```
SSELECT ACC-PAYABLE BY NAME WITH AMOUNT "100" NAME ACCOUNT AMOUNT
```

This command will generated a selected list containing the following information:

```
001 Boy George,
002 Voice Lesson
003 $4,523.59.
004 Eddie Van Halen,
005 Guitar String
006 $123.45.
007 Magic Johnson,
008 Basketball Shoes
009 $250.00.
```

Note that the correlatives on the names and on the amounts have been performed. Now by issuing the following RUNOFF command the form letters are generated:

```
RUNOFF DICT ACC-PAYABLE LETTER (P)
```

The form letters will be printed as follows:

Dear Mr. Boy George,

Our records show that your Voice Lesson account is overdrawn by the amount of $4,523.59. We would appreciate prompt payment.

Thank You,

Indiana Jones

President CELEBRITY SERVICES CO.

(next page)

Dear Mr. Eddie Van Halen,

Our records show that your Guitar String account is overdrawn by the amount of $123.45. We would appreciate prompt payment.

Thank You,

Indiana Jones

President CELEBRITY SERVICES CO.

(next page)

Dear Mr. Magic Johnson,

Our records show that your Basketball Shoes account is overdrawn by the amount of $250.00. We would appreciate prompt payment.

Thank You,

Indiana Jones

President CELEBRITY SERVICES CO.

8.3.37 SAVE INDEX file-name

This command causes chapter and page number information of indexed data in a text to be saved in a separate file. Each indexed datum is stored as an individual item using the datum as the Item-Id, the chapter (where that datum is referenced) as the first attribute and the page number as the second attribute. Multiple values are stored in these attributes as multiple references to the same indexed datum are encountered. The resulting file may then be operated on by the ACCESS processor to generate listings for the chapter and page number information of all indexed data in a text.

The 'file-name' is the name of the file in which the chapter and page information is to be stored. NOTE: This must be a SEPARATE FILE from the text file !!! (Otherwise data in the text file will be DESTROYED.) The .SAVE INDEX command is placed in the text item itself and must precede the '.INDEX' commands. In short, only that indexed data which has been preceded by the '.SAVE INDEX' command will be saved in the specified file.

8.3.38 SECTION n text

This command may be used in conjunction with the CHAPTER command to handle automatic chapter section numbering and formattin. The SECTION command automatically starts the next section at depth n, where n is the range 1-5. The text is printed following the section number and SKIP occurs. The text is recorded as the section heading in the TABLE OF CONTENTS. If no text appears on the SECTION command, then no SKIP occurs and the section is not recorded in the TABLE OF CONTENTS. Section numbers are incremented automatically and the section number is printed in the form i.j.k.l.m with n digits printed.

Conventionally the .SECTION command is followed by a blank line before the next paragraph starts. Since the SECTION command causes a break which terminates the preceding paragraph, and since the text following the SECTION command is placed immediately into an ouput line and output prior to a consideration of the next line, the blank line after the SECTION command can be avoided by not indenting the first line of the next paragraph. That is, if the processor does not know that the next line starts a paragraph, it will not skip a line. It may be necessary to use an INDENT MARGIN if paragraph indentation is desired, however.

8.3.39 SET TABS n,n,n, ...

This command clears previous tab stops and sets new tab stops as indicated by the numeric tab positions. The tab stops (up to 30) must be greater than zero and in increasing order. They indicate tab stop positions relative to the left margin. Tabs are only in effect in NOFILL mode. The left-tab character (<) causes the next word to start at the next tab position. The right-tab character (>) causes the next word to end at the next tab position. If a tab character appears at a point in the line where no further tab stops have been set, the tab character is ignored.

8.3.40 SKIP n (SK)

The SKIP causes a BREAK after which n*(SPACING n) lines are left blank. If the skip would advance past the end of the page, the output is advanced to the top of the next page. If n is missing, n=1 is assumed.

8.3.41 SPACE n (SP)

This command has the same affect as SKIP, except that n (rather than N*(SPACING n) lines are left blank. SPACE is used where space is to be left independent of the line spacing; SKIP is used where space should be relative to the SPACING command. If n is missing, n=1 is assumed.

8.3.42 SPACING n

This command sets the line spacing to n. The command .SPACING 2 may be used for double spacing.

8.3.43 STANDARD

This command sets the standard (default) parameters and modes. The STANDARD command is equivalent to the following commands:

```
.CS.F.J.UC.LEFT MARGIN 0.CRT.HEADING
.FOOTING
.PARAGRAPH 5.LINE LENGTH 70
```

8.3.44 TEST PAGE n

This command causes a BREAK followed by an advance to a new page when there are less than n lines remaining on the current page. If there are n or more lines remaining on the current page, this command has no effect. This command should be used to ensure that the following n lines are all output on the same page.

8.3.45 UPPER CASE (UC)

This command puts RUNOFF into upper case mode. Alphabetic letters will be processed as they are, unless modified by special commands or control characters. This command allows users of terminals with upper and lower case to generate the input text file without special commands or control characters. UC is one of the STANDARD settings. (See the STANDARD command.)

The 'U' option will force the whole runoff output to upper-case if that is desired.

RUNOFF features Special Control Characters for Underlining, Tab Setting, Upper/Lower Case, and Special Character Override.

8.4.1 Upper- and lower-case controls.

The upper-case, lower-case control structure causes the text to go to the case specified. ENDCASE or EC will turn off both the upper-case condition and the lower-case condition to allow the text to go to its natural condition.

The forms ^^ and \\ cause the text to switch to upper-case or to lower-case in the same way that UC and LC cause the switch, except that ^^ and \\ may be imbedded in a line. Turning off the condition ^^ or \\ requires the use of EC.

The forms ^, \, &, and @ will produce one character of upper-case, lower-case, underline, or overstrike. Each will be treated as the character itself if it is followed by a blank. The backarrow or underline character, _, will cause the succeeding character to be taken as a text character rather than a control character. This means that if you have existing RUNOFF text with forms such as '40# @$1.28/#', the dollar sign will be overstruck and the @ will disappear unless the backarrow, _, is inserted in front of the @. The same is true of the '&' character if it occurs in a character string.

The example below is an attempt to display the interactions of the several commands above. The first part is the text which was sent to RUNOFF and the second part is the output from RUNOFF. First, note that the 'I' in 'is' is always capitalized by the single-character ^, and that the 'a' is always in lower-case due to the single-character \ command.

The first line is in its natural form. The second line is uniformly capitalized by the UC command, excepting the 'a'. The third line is uniformly sent to lower-case, except for the 'Is'. The fourth and fifth lines contain a '^^ text \\' string, which is uniformly capitalized, excepting the 'a'. After the \\ the string reverts to lower-case. The only way to retrieve the capitalization of the string 'UC AND 'LC' is by the use of EC command. Thus, the sixth line is in its natural form.

```
.LINE LENGTH 66.PARAGRAPH 5.J
 This ^is \a test of UC AND LC.
.uc
 This ^is \a test of UC AND LC.
.lc
 This ^is \a test of UC AND LC.
 This ^is \a ^^test of UC AND LC.
 This ^is \a test\\ of UC AND LC.
.ec
 This ^is \a test of UC AND LC.


 This Is a test of UC AND LC.

 THIS IS a TEST OF UC AND LC.

 This Is a test of uc and lc.

 This Is a TEST OF UC AND LC.

 THIS IS a TEST of uc and lc.

 This Is a test of UC AND LC.
```

Example of .UC, .LC, .EC and the associated ^ and \ characters.

8.4.2 Underlining and overstriking.

UNDERLINING The ampersand (&) may be used to indicate underlining. The ampersand causes the letter immediately following to be underlined.

```
.LC
THE LETTER &A IS FIRST IN THE ALPHABET
```

This example of RUNOFF source would print as:

```
he letter a is first in the alphabet
```

Ampersand may be used in conjunction with the up-arrow and back-slash to underline a series of characters. An ampersand followed immediately by an up-arrow (&^) puts RUNOFF in the Underline mode until an ampersand followed immediately by a back-slash (&\) is encountered.

```
.UC
&^SPECIAL CONTROL CHARACTERS&\ ARE NEEDED ...
```

This example of RUNOFF source would print as:

```
SPECIAL CONTROL CHARACTERS ARE NEEDED ...
```

The S option suppresses underlings and overstriking.

8.4.3 Tab setting.

TAB SETTINGS The less-than (<) and greater-than (>) characters may be used for tabbing. The left-tab character (<) causes the next word to start at the next tab position as set by the .SET TABS command. The right-tab character (>) causes the next word to end at the next tab position.

```
.NF
.SET TABS 5,8,25
.SK 1
><&^NAME<CONVENTIONAL DATA PROCESSING NAME&\
.SK 1
>1.<Item<Record
>1a.<Attribute<Field
>1b.<Item-id<Record Key
```

This example of RUNOFF source would print as:

```
         NAME              CONVENTIONAL DATA PROCESSING NAME

     1.  Item              Record
    1a.  Attribute         Field
    1b.  Item-id           Record Key
```

Note: Tab characters are only in effect in the NOFILL (NF) mode. You will also note that the sequence <<< will tab over to the third tab if tabs are set, and that the right tabs are more cooperative.

BOLDFACE PRINTING The at sign (@) may be used to indicate BOLDFACE type. An at sign followed immediately by an up-arrow (@^) puts RUNOFF in the boldface mode until an at sign followed immediately by a back-slash (@\) is encountered. The number of times the boldface letters are overprinted may be set by using the numeric option of the RUNOFF verb.

```
.UC
@^SPECIAL CONTROL CHARACTERS@\ ARE NEEDED ...
```

This example of RUNOFF source would print as:

```
SPECIAL CONTROL CHARACTERS ARE NEEDED ...   SPECIAL CONTROL CHARACTERS
```

SPECIAL CHARACTER OVERRIDE The back-arrow (_) may be used to quote one of the special control characters or blanks. the letter immediately following the back-arrow is transmitted to the output without special processing.

```
_^SPECIAL _^CONTROL _^CHARACTERS ARE NEEDED ...
```

This example of RUNOFF source would print as:

```
^SPECIAL ^CONTROL ^CHARACTERS ARE NEEDED ...
```

SECTION 9

THE ICON/PICK PICK/BASIC LANGUAGE

ICON®

Chapter 9

PICK/BASIC

THE PICK SYSTEM

USER MANUAL

This manual describes the PICK/BASIC source language, which is an extended version of Dartmouth BASIC.

BASIC (Beginners All-Purpose Symbolic Instruction Code) is a simple yet versatile programming language suitable for expressing a wide range of problems. Developed at Dartmouth College in 1963, BASIC is a language especially easy for the beginning programmer to master. Extended PICK/BASIC has the following extraordinary features:

- Optional statement labels (i.e., statement numbers)
- Statement labels of any length
- Multiple statements on one line
- Computed GOTO statements
- Complex IF statements
- Multi-line IF statements
- Priority case statement selection
- String handling with variable length strings up to 32,267 characters
- External subroutine calls
- Direct and indirect calls
- Magnetic tape input and output
- Fixed point arithmetic with up to 15 digit precision
- ACCESS data conversion capabilities
- PICK file access and update capabilities
- File level or group level lock capabilities
- Pattern matching
- Dynamic arrays

ABORT	FOOTING	MATCHES	PROMPT	SLEEP
BREAK ON/OFF	FOR...NEXT	MATREAD	READ	STOP
CALL	GO	MATREADU	READNEXT	SUBROUTINE
CASE	GOSUB	MATWRITE	READV	UNLOCK
CHAIN	GOTO	MATWRITEU	READT	WEOF
CLEAR	GO TO	NEXT	READV	WRITE
CLEARFILE	HEADING	NULL	READVU	WRITEU
COMMON	IF	ON GOSUB	RELEASE	WRITET
DATA	INPUT	ON GOTO	REM * !	WRITEV
DELETE	INPUTERR	OPEN	RETURN	WRITEVU
DIM	INPUTNULL	PAGE	RETURN TO	
END	INPUTTRAP	PRECISION	REWIND	
ENTER	INPUT @	PRINT	RQM	
EQUATE	MAT	SELECT	PRINTER ON/OFF	

PICK/BASIC Statements

@	COUNT	ICONV	NUM	SPACE
ABS	DATE ()	INDEX	OCONV	SQRT
ALPHA	DCOUNT	INSERT	PWR	STR
ASCII	DELETE	INT	REM	TAN
CHAR	EBCDIC	LEN	REPLACE	TIME ()
COL1	EXP	LN	RND	TIMEDATE ()
COL2	EXTRACT	LOCATE	SEQ	TRIM
COS	FIELD	NOT	SIN	

PICK/BASIC Intrinsic Functions

A PICK/BASIC program is comprised of PICK/BASIC statements. PICK/BASIC statements may contain variables, constants, expressions, and PICK/BASIC Intrinsic Functions.

A PICK/BASIC program consists of a sequence of PICK/BASIC statements. More than one statement may appear on the same program line, separated by semicolons. Any PICK/BASIC statement may begin with an optional statement label.

PICK/BASIC statements may contain arithmetic, relational, and logical expressions. These expressions are formed by combining specific operators with variables, constants, or PICK/BASIC Intrinsic Functions. The value of a variable may change dynamically throughout the execution of the program. A constant, as its name implies, has the same value throughout the execution of the program. An Intrinsic Function performs a pre-defined operation upon the parameter(s) supplied.

FUNCTION	BRIEF DESCRIPTION
ABS	Returns an absolute value.
ALPHA	Tests for alphabetic value.
ASCII	Converts string from EBCDIC to ASCII.
CHAR	Converts numeric value to ASCII character.
COL1	Returns column position preceding FIELD-selected sub-string
COL2	Returns column position following FIELD-selected sub-string
DATE	Returns current internal date.
DELETE	Deletes attribute, value, or sub-value from dynamic array.
EBCDIC	Coverts string from ASCII to EBCDIC.
EXTRACT	Returns attribute, value, or sub-value from dynamic array.
FIELD	Returns a delimited sub-string.
ICONV	Provides for Pick input conversion.
INDEX	Returns column position of sub-string.
INSERT	Inserts attribute, value, or sub-value into dynamic array.
INT	Return an integer value.
LEN	Returns length of string.
LOCATE	Returns the index of a sub-string in a dynamic array.
NOT	Returns logical inverse.
NUM	Tests for numeric value.
OCONV	Provides for Pick output conversion.
REPLACE	Replaces attribute, value, or sub-value in dynamic array.
RND	Generates random number.
SPACE	Generates string containing blanks.
STR	Generates specified string.
TIME	Returns internal time of day.
TIMEDATE	Returns external time and date.
Controls terminal cursor.	

Summary of PICK/BASIC INTRINSIC FUNCTIONS

STATEMENT	BRIEF DESCRIPTION
BREAK ON/OFF	Enables or disables debugger.
CALL	External subroutine branching.
CASE	Provides conditional selection of a sequence of statements.
CHAIN	Passes control to another program.
CLEAR	Initializes all variables to zero.
CLEARFILE	Clears data section of specified file.
COMMON	Variable storage space allocation, used with CHAINed programs.
DELETE	Deletes specified file item.
DIM	Reserves storage for arrays.
END	Designates the physical end of the program.
EQUATE	Allows variable to be defined as equivalent of another.
FOR...NEXT	Specifies beginning of a program loop, NEXT specifies end.
GOSUB	Transfers control to a subroutine.
GOTO	Transfers control to another statement.
HEADING	Prints a page heading.
IF	Provides conditional execution of specified statements.
INPUT	Inputs data from the terminal.
LOCK	Sets an execution lock.
LOOP...REPEAT	Provides for structured program loops.
MAT	Assigns value to each element of an array.
MATREAD	Reads a file item into an array.
MATREADU	Reads a file item into an array, sets update lock.
MATWRITE	Writes a file item with the contents of an array.
MATWRITEU	Same as MATWRITE but will not unlock update group.
NULL	Specifies a non-opertion.
OPEN	Selects a file for subsequent I/O.
PAGE	Pages output device and prints heading.
PRINT	Causes specified data to be printed.
PRINTER ON/OFF	Controls selection of printer or terminal for output.
PROMPT	Selects a prompt character for the terminal.
READ	Reads a file item.
READU	Reads a file item, sets update lock.
READNEXT	Reads next item-id.
READT	Reads next magnetic tape record.
READV	Reads an attribute.
READVU	Reads an attribute, sets update lock.
REM	Specifies a remark (command) statement.
RETURN	Returns control from a subroutine.
REWIND	Rewinds magnetic tape.
RQM or SLEEP	Terminates programs current time quantum.
STOP	Designates a logical end of the program.
SUBROUTINE	Specifies a program branch subroutine.
UNLOCK	Resets an execution lock.
WEOF	Writes an EOF on magnetic tape.
WRITE	Updates a file item.
WRITET	Writes a magnetic tape record.
WRITEU	Writes a file item, will not unlock update group.
WRITEV	Updates an attribute value.
WRITEVU	Updates an attribute value, will not unlock update group.

Summary of PICK/BASIC Statements

A PICK/BASIC program, when stored, constitutes a File Item, and is referenced by its Item-Name (i.e., the name it is given when it is created via the EDITOR. An individual line within the PICK/BASIC program constitutes an attribute.

There is a fixed structure for PICK/BASIC source files. The file MUST have a dictionary and a separate data level. The PICK/BASIC source programs are stored in the data level of the file. The compiler writes the object and the symbol file as one record into the dictionary. This makes it much simpler to manipulate the program source. It can be LISTed, T-DUMPed, T-LOADed, and so on, without having to select the source items. The object record has the same format as a pointer-file record and so the dictionary "D" pointer must have a "DC" in attribute one. The primary advantages of this new format are:

1. The object can now be protected with access/update locks.

2. The object saves/restores with the account on account-saves.

3. The CATALOG function is not necessary for run time efficiency.

4. There is less disk space utilized and fewer steps to perform.

5. The PICK/BASIC Debugger can tell the name of the item and verify the object code integrity.

A PICK/BASIC program is comprised of PICK/BASIC statements. The Remark statement may be used to identify the function or purpose of various sections of the program.

SEMICOLON - ';'

A PICK/BASIC program consists of a sequence of PICK/BASIC statements. More than one statement may appear on the same program line, separated by semicolons. For example:

```
X = 0;  Y = 0;  GOTO 50
```

LABELS - optional

Any PICK/BASIC statement may begin with an optional statement label. A statement label is used so that the statement may be referenced from other parts of the program. A statement label may be any constant whole number. The following INPUT statement, for example, has a statement label of 100:

```
100 INPUT X
```

Statement labels may be included or omitted at the programmer's option.

REMARKS - 'REM' '!' '*'

A helpful feature to use when writing a PICK/BASIC program is the Remark statement. A Remark statement is used to explain or document the program. It allows the programmer to place comments anywhere in the program without affecting program execution. A Remark statement is specified by typing the leters REM, or the asterisk character (*), or the exclamation (!) at the beginning of the statement; any arbitrary characters may then follow (up to the end of the line). For example:

```
REM THESE PICK/BASIC STATEMENTS
!   DO NOT AFFECT
*   PROGRAM EXECUTION
```

BLANK SPACES

Except for situations explicitly called out in the following sections, blank spaces appearing in the program line (which are not part of a data item) will be ignored. Thus, blanks may be used freely within the program for purposes of appearance.

```
REM  PROGRAM TO PRINT THE
*    NUMBERS FROM ONE TO TEN
*
     I = 1; * START WITH ONE
5    PRINT I; * PRINT THE VALUE
     IF I = 10 THEN STOP; * STOP IF FINISHED
     I = I + 1; * INCREMENT I
     GOTO 5; * START OVER
     END
```

Sample PICK/BASIC Program Including Remark Statements.

PICK/BASIC allows the user to manipulate PICK file items in the form of item-formatted strings which are called dynamic arrays.

The PICK/BASIC language contains a number of statements and functions which are extremely useful in accessing and updating PICK files. A complete description of the PICK file structure is presented in the chapter on File-structure. A brief description of the structure as viewed by the PICK/BASIC programmer is appropriate at this point.

A PICK file consists of a set of items. When a PICK file item is accessed by a PICK/BASIC program (refer to INPUT/OUTPUT STATEMENTS), it is represented as a PICK/BASIC string in item format. A string in item format is called a dynamic array.

A dynamic array consists of one or more attributes separated by attribute marks (i.e., an attribute mark has an ASCII equivalent of 254, which prints as "^"). An attribute, in turn, may consist of a number of values separated by value marks (i.e, a value mark has an ASCII equivalent of 253, which prints as "]"). Finally, a value may consist of a number of secondary values separated by secondary value marks (i.e., a secondary value mark has an ASCII equivalent of 252, which prints as "\"). An example of a dynamic array is as follows:

"55^ABCD^732XYZ^100000.3#"

where "55", "ABCD", "73XYZ", and "100000.33" are attributes.
The following illustrates a more complex dynamic array:

"Q5^AAAA^952]ABC]12345^A^B^C]TEST\12I\9\99.3]2^555"

where "Q5", "AAAA", "952]ABC]12345", "A", "B", "C]TEST]\12I\9\99.3]2" and "555" are attributes; "952", "ABC", "12345", "C", "TEST\12I\9\99.3", and "2" are values; and "TEST", "12I", "9", and "99.3" are secondary values.

Dynamic arrays can be directly manipulated by the PICK/BASIC dynamic array functions (refer to the section titled PICK/BASIC INTRINSIC FUNCTIONS). Dynamic arrays are called "arrays" because they can be referenced by these functions using 3 subscripts. They are "dynamic" in the sense that elements can be added and deleted without having to re-compile the program, as long as the item does not exceed 32,267 characters.

ARRAY	EXPLANATION
123^456^789]ABC]DEF	"123", "456", "789]ABC]DEF" are attributes; "789", "ABC" and "DEF" are values.
1234567890	"1234567890" is an attribute.
Q56^3.22]3.56\88\B2C^99	"Q56", "3.22]3.56\88\B]C", and "99" are attributes; "3.22", "3.5688b", and "C" are values; "3.56", "88", and "B" are secondary values.
A]B]C]D^E]F]G]H^I]J	"A]B]C]D", "E]F]G]H", and "I]J" are attributes; "A", "B", "C", "D", "E", "F", "G", "H", "I", and "J" are values.

Sample Usage of Dynamic Arrays.

9.5 CREATING AND COMPILING PICK/BASIC PROGRAMS

A PICK/BASIC program is created via the EDITOR as any other data-file item. Once this source code item has been filed, it is compiled by issuing the COMPILE verb (or the PICK/BASIC verb) at the TCL level.

FORMAT:

ED (or EDIT) file-name item-id

Upon execution, the EDITOR processor will then be entered, and the user may begin entering his PICK/BASIC program. For ease of instruction indentation, the user may set tab stops (either at the TCL level or while the EDITOR processor is in control-- see examples below).

The program name will be that specified by the 'item-id' and the program will be stored in the file specified by the 'file-name'. Users will typically have a file exclusively devoted to the storage of PICK/BASIC programs. The PICK/BASIC compiler stores the object code in the same file, but not the same item, as the source code (see below).

Once the PICK/BASIC program has been entered and filed, it may be compiled by issuing the BASIC verb at the TCL level. BASIC is a TCL-II verb which creates a new file item: it contains the compiled PICK/BASIC program (the object code), and a symbol definition table of the variables used in the program. The object code item has an item-id of a '$' concatenated with the item-id Both items are stored in the same file (specified by 'file-name').

FORMAT:

BASIC file-name item-list {(options}

The 'item-list' consists of one or more item-id's (program names) separated by one or more blanks. The 'options' parameter is optional and if used, must be preceded by a left parentheses. Multiple options should be separated by commas. Valid options are listed below. For detailed descriptions of each, see following section.

BASIC VERB OPTIONS

A	Assembled code option
C	Suppress End Of Line (EOL) opcodes from object code.
E	List error lines only.
L	List PICK/BASIC program.
M	List map of PICK/BASIC program.
P	Print compilation output on line printer.
S	Suppress generation of symbol table.
X	Cross reference all variables.


```
* >TABS I 4,8,12  [CR]  <------------ User sets input tabs
                                      at TCL level

* >ED BP COUNT  [CR]    <------------ User edits item 'COUNT'
                                      in file 'BP' (Basic Programs)
 NEW ITEM
 TOP
*  .I  [CR]             <------------ User enters input mode and
                                      begins to enter program

* 001* PROGRAM COUNTS FROM 1-10 *  [CR]
* 002   FOR I = 1 TO 10  [CR] <----- Entered with [C] I (or TAB key)
* 003     PRINT I  [CR]  <-------     depressed once for indentation
* 004   NEXT I  [CR]                  to first tab stop.
* 005 END  [CR]                       |
* 006  [CR]                          ----- [C] I (or TAB key) depressed
TOP                                         twice for second tab stop
                                            indentation
* .FI  [CR]          <---------
                                  |
                                  -------- User files item
'COUNT' FILED

* >BASIC BP COUNT  [CR] <---------- User issues compile command
  *****
 [B0] LINE 5 COMPILATION COMPLETED
```

PICK/BASIC Program "COUNT" Created (edited), Filed and Compiled.

9.6 PICK/BASIC COMPILER OPTIONS: A, C, E, L AND P OPTIONS

This section describes five of the options available when issuing the ' BASIC ' compile statement. They are the "A" for assembled code, the "C" for suppression of end of line opcode, "E" for the listing of error lines only, the "L" for the listing of the program during compilation, and "P" for routing output to the printer. The following section describes the remaining three compiler options.

FORMAT:

BASIC file-name item-name {(options}

If multiple options are present, they are seperated by commas.

A - The assembled code option. The "A" option generates a listing of the source code line numbers, the labels and the PICK/BASIC opcodes used by the program. This is a 'pseudo' assembly code listing which allows the user to see what PICK/BASIC opcodes his program has generated. The hexadecimal numbers on the left of the listing are the PICK/BASIC opcodes and the mnemonics are listed on the right. The assembled code listing of the PICK/BASIC program "COUNT" (from previous section) is shown, as an example, on the facing page.

C - The catalog option. The catalog option suppresses the end-of-line (EOL) opcodes from the object code item. The EOL opcodes are used to count lines for error messages. This eliminates 1 byte from the run time object code for every line in the source code. This option is designed to be used with debugged cataloged programs. Any run time error message will specify a line number of 1.

E - The 'list error lines only' option. The "E" option generates a listing of the error lines encountered during the compilation of the program. The listing indicates line number in the source code item, the source line itself and a description of the error associated with the line.

L - The list program option. The "L" option generates a line by line listing of the program during compilation. Error lines with associated error messages are indicated.

P - The printer option. The "P" option routes all output generated by the compilation to the printer.


```
SOURCE     BASIC      PSEUDO
CODE       OBJECT     ASSEMBLY
LINE NO.   CODE       CODE

001        01         EOL
002        03         LOADA      I
002        FD         LOAD.      1
002        20         ONE
002        2D         SUBTRACT
002        5F         STORE
002           *1001
002        05         LOADN      10
002        03         LOADA      I
002        20         ONE
002        28         FORTEST    *2001
002        01         EOL
003        5D         LOAD       I
003        50         PRINTCRLF
003        01         EOL
004        06         BRANCH     *1001
004           *2001
004        01         EOL
005        01         EOL
006        45         EXIT

[B0] LINE 6 COMPILATION COMPLETED
```

"A" option listing of PICK/BASIC program "COUNT"

This section describes the remaining three options available when issuing the BASIC compile statement. They are the "M" for map, the "S" for suppressing generation of the symbol table, and the "X" for cross reference.

M - The map option. The "M" option generates a variable map and a statement map, both of which are printed out after compilation. These maps show where the program data has been stored in the user's workspace. The variable map lists the offset in decimal (from the beginning of the seventh frame of the IS buffer) of every PICK/BASIC variable in the program. For example, the form:

```
20 xxx 30 yyy
```

shows that the descriptor of variable 'xxx' starts on byte 20 and the descriptor of variable 'yyy' starts on byte 30 of the seventh frame of the IS buffer. Descriptors are 10 bytes in length.

The statement map shows which statements of the PICK/BASIC program are contained in the frames of the OS buffer. If the program is 'run' frame number '01' refers to the seventh frame of the OS buffer. If the program is cataloged, frame 01 will be specified in the catalog pointer item in the POINTER-FILE. The statement map may be used to determine if frequently executed loops cross frame boundaries.

S - The suppress symbol table option. The "S" option suppresses the the symbol table item which is normally generated during compilation.

The symbol table item is used exclusively by the PICK/BASIC DEBUGGER for reference, therefore it must be kept on file only if the user wishes to use the Debugger.

X - The cross reference option. The "X" option creates a cross reference of all the labels and variables used in a PICK/BASIC program and stores this information in the BSYM file. Note: A BSYM file must exist (a modulo and separation of 1,1 should be sufficient). The "X" option first clears the BSYM file information in the BSYM file then creates an item for every variable and label used in the program. The item-id is the variable or label name. The attributes contain the line numbers of where the variable or label is referenced. An asterisk will precede the line number where a label is defined, or where the value of the variable is changed.

No output is generated by this option. An attribute definition item should be placed in the dictionary of the "BSYM" file which allows a cross reference listing of the program to be generated by the command: >SORT BSYM BY LINE-NUMBER LINE-NUMBER.

9.8 EXECUTING PICK/BASIC PROGRAMS

| The PICK/BASIC program is executed by issuing the RUN verb. |

FORMAT:

RUN file-neme item-id {(options)}

RUN is the TCL-II verb issued to run a compiled PICK/BASIC program. The "file-name" and "item-id" specify the compiled PICK/BASIC program to be executed. The "options" parameter is optional (if used, it must be enclosed in parentheses). Multiple options are separated by commas. Valid options are as follows:

A - Abort option. The "A" option inhibits entry to the Basic Debugger under all error conditions; instead, the program will print a message and terminate execution.

D - Run-time debug option; causes the PICK/BASIC debugger to be entered before the start of program execution. Note that the PICK/BASIC debugger may also be called at any time while the program is executing, by pressing the BREAK key on the terminal.

E - Errors option. The "E" option forces the PICK/BASIC runtime package to enter the PICK/BASIC Debugger whenever an error condition occurs. The use of this option will force the operator to either accept the error by using the Debugger, or exit to TCL.

I - Inhibit initialization of data area. The "I" option should only be used in the "CHAIN" statement. (refer to the description of the PICK/BASIC CHAIN statement).

N - Nopage option. The "N" option cancels the default wait at the end of each page of output.

P - Printer on (has same effect as issuing a PICK/BASIC PRINTER ON statement). Directs all program output to the printer.

S - Suppress run-time warning messages.

```
* >RUN PROGRAMS TESTING  [CR]
THIS IS
A TEST

>
```

Execution of Sample PICK/BASIC Program

PICK/BASIC programs may be shared by cataloguing and may be used in PROC, as outlined in this topic.

PROGRAM SHARING

A PICK/BASIC program may be shared among users by loading its object code into system disk space and evoking it via a verb at the TCL level. To load the program into the system disk space, issue the CATALOG verb.

FORMAT:

```
CATALOG file-name item-id
```

The "file-name" and "item-id" specify the previously compiled PICK/BASIC program which is to be loaded. If there are no conflicts, the system will respond with:

```
[241] item-id CATALOGED;  x FRAMES USED.
```

where "x" is the size of the object code in frames (500 bytes each). Once a program is cataloged, it is 'run' simply by issuing the program name at the TCL prompt. Do not use the RUN verb! The TCL-II verb which is added to the user's Master Dictionary (if not already present) has the following form:

```
1)  P
2)  10B4
3)
4)
5)  XXXXX
```

where XXXXX is the user's account name. If an item already exists in the user's Master Dictionary which is not in the above form, the system will respond with:

```
[415] item-id EXISTS ON FILE
```

and the program will not be cataloged.

In order to decatalog a PICK/BASIC program, the following verb has been provided.

FORMAT:

```
DECATALOG item-id {account-name}
```

An item is maintained in the POINTER-FILE for each cataloged PICK/BASIC program. The DECATALOG statement will delete that item from the POINTER-FILE, returning all of the overflow space, and deleting the verb from the user's Master Dictionary. After deletion, verbs executed from other dictionaries will receive the message:

```
'item-id' NOT ON FILE
```


9.10 PICK/BASIC EXECUTION FROM PROC

PICK/BASIC program execution can be intiated from PROC, similar to any other TCL command.

A PICK/BASIC program may be run from a PROC. The following example illustrates the use of a PICK/BASIC program in conjunction with the ACCESS Sort Select (SSELECT) verb.

PROC named LISTBT as follows:

```
PQ
HSSELECT BASIC/TEST
STON
HRUN BASIC/TEST LISTIDS
P
```

PICK/BASIC program named LISTIDS as follows:

```
     OPEN '','BASIC/TEST' ELSE PRINT 'FILE MISSING'; STOP
10   N = 0
20   READNEXT ID ELSE STOP
     PRINT ID 'L##################':
     N = N + 1
     IF N>= 4 THEN PRINT; GO TO 10
     GO TO 20
     END
```

By typing in the following:

```
LISTBT
```

at the TCL level, the PROC LISTBT selects the item-id's contained in file BASIC/TEST and invokes the BASIC program LISTIDS to list the item-id's selected, four to a line.

Sample Usage of PICK/BASIC called from PROC.

There are two types of data: NUMERIC and STRING. These data types are represented within the PICK/BASIC program as either variables or as constants.

Numeric data consists of a series of digits and represent an amount (e.g., 255). String data consist of a set of characters, such as would be for a name and address. For example:

```
JOE DOE, 430 MAIN, ATOWN, CA.
```

These data types may be represented within the PICK/BASIC program as either constants or variables. A constant, as its name implies, has the same value throughout the execution of the program. A numeric constant may contain up to 15 digits, including a maximum of 4 digits following the decimal point and must be in the range:

```
-14,073,748,835.0000 to 14,073,748,835.0000
```

if the PRECISION (see section on PRECISION DECLARATION) of the program is 4 digits; by setting the PRECISION to a value less than 4, the range of the allowable numbers is increased accordingly.

The unary minus sign is used to specify negative constants. For example:

```
-17000000
-14.3375
```

A string constant is represented by a set of characters enclosed in single quotes, double quotes, or backslashes. For example:

```
"THIS IS A STRING" 'ABCD1234#*' \HELLO\
```

if any of the string delimiters (',", or \) are to be part of the string, then one of the other delimiters must be used to delimit the string. For example:

```
"THIS IS A 'STRING' EXAMPLE"
\THIS IS A "STRING" EXAMPLE\
```

A string may contain from 0 to 32,267 characters (i.e., maximum length of a PICK file item).

As mentioned above, data may also be represented as variables. A variable has a name and a value. The value of a variable may be either numeric or string, and may change dynamically throughout the execution of the program. The name of a variable identifies the variable (the name remains constant throughout program execution). Variable names consist of an alphabetic character followed by zero or more letters, numerals, periods, or dollar signs. The length of a variable name may be from 1 to 64 characters.

For example:

```
X
QUANTITY
DATA.LENGTH
B$..$
```

The variable X, for example, may be assigned the value 100 at the start of a program, and may then later be assigned the value "THIS IS A STRING".

It should be noted that PICK/BASIC keywords (i.e., words that define PICK/BASIC statements and functions) may not be used as variable names.

VALID STRING	INVALID STRING
"ABC%123#*4AB"	ABC123 (i.e., quotes are missing)
'1Q2Z....'	
"A 'LITERAL' STRING"	'ABC%QQR" (i.e., either two single quotes or two double quotes must be used)
'A "LITERAL" STRING'	
'' (i.e., the empty string)	"12345678910 (i.e., terminating double quote missing)
\JOHN PROGRAMMER\	

Sample Usage of String Constants

VALID VARIABLE NAME	INVALID VARIABLE NAME
A5	ABC 123 (i.e., no space allowed)
ABCDEFGHI	
QUANTITY.ON.HAND	5AB (i.e., must begin with letter)
R$$$$P$	Z.,$ (i.e., comma not allowed)
J1B2Z	
INTEGER	A-B (i.e., "-" not allowed)
THIS.IS.A.NAME	

Sample Usage of Variable Names

9.12 ARITHMETIC EXPRESSIONS

Expressions are formed by combining operators with variables, constants, or PICK/BASIC Intrinsic Functions. Arithmetic expressions are formed by using arithmetic operators.

When an expression is encountered as part of a PICK/BASIC program statement, it is evaluated by performing the operations specified by each of the operators on the adjacent operands, i.e., the adjacent constants, identifiers, or Intrinsic Functions. (NOTE: Intrinsic Functions are discussed in a separate section of this manual.)

Arithmetic expressions are formed by using the arithmetic operators listed below. The simplest arithmetic expression is a single numeric constant, variable, or Intrinsic Function. A simple arithmetic expression may combine two operands using an arithmetic operator. More complicated arithmetic expressions are formed by combining simple expressions using arithmetic operators.

When more than one operator appears in an expression, certain rules are followed to determine which operation is to be performed first. Each operator has a precedence rating. In any given expression the highest precedence operation will be performed first. Precedence of the arithmetic operators are shown below. If there are two or more operators with the same precedence (or an operator appears more than once) the leftmost operation is performed first. For example, consider this expression: -R/A+B*C. The unary minus is evaluated first (i.e., -R = result 1). The expression then becomes: result 1 / A+B*C. The division and multiplication operators have the same precedence; since the division operator is leftmost it is evaluated next (i.e., result 1 / A = result 2). The expression then becomes: result 2+B*C. The multiplication operation is performed next (i.e., B*C = result 3). The result 2 + result 3 = Final Result.

Using some figures in the above expression illustrates, for example, that the expression -50/5+3*2 evaluates to -4.

Any sub-expression may be enclosed in parentheses. Within the parentheses, the rules of precedence apply. However, the parenthesized subexpression as a whole has highest precedence and is evaluated first. For example: (10+2)*(3-1) = 12*2 = 24. Parentheses may be used anywhere to clarify the order of evaluation, even if they do not change the order.

If a string value containing only numeric characters is used in an arithmetic expression, it is considered as a decimal number. For example, 123 + "456" evaluates to 579.

If a string value containing non-numeric characters is used in an arithmetic expression, a warning message will be printed (refer to APPENDIX D - PICK/BASIC RUN-TIME ERROR MESSAGES) and zero will be assumed for the string value.

The following expression, for example, evaluates to 123:

```
123 + "ABC"
```


OPERATOR SYMBOL	OPERATION	PRECEDENCE
+	unary plus	1 (high)
-	unary minus	
*	multiplication	2
/	division	2
+	addition	3
-	subtraction	3 (low)

Arithmetic Operators

2+6+8/2+6	Evaluates to 18
12/2*3	Evaluates to 18
12/(2*3)	Evaluates to 2
12/2*3	Evaluates to 18
12/(2*3)	Evaluates to 2
A+75/25	Evaluates to 3 plus the current value of variable A.
-5+2	Evaluates to -3
-(5+2)	Evaluates to -7
8*(-2)	Evaluates to -16
5 * "3"	Evaluates to 15

Sample Usage of Arithmetic Expressions.

A string expression may be any of the following: a string constant, a variable with a string value, a sub-string, or a concatenation of string expressions. String expressions may be combined with arithmetic expressions.

FORMAT:

variable [expression1,expression2]

expression1	starting character position
expression2	number of characters in sub-string length

A sub-string is a set of characters which makes up part of a whole string. For example, "SO.", "123", and "ST." are sub-strings of the string "1234 SO. MAIN ST." Sub-strings are specified by a starting character position and a sub-string length, separated by a comma and enclosed in square brackets. For example, if the current value of variable S is the string "ABCDEFG", then the current value of S[3,2] is the sub-string "CD" (i.e., the two character sub-string starting at character position 3 of string S). Furthermore, the value of S[1,1] would be "A", and the value of S[2,6] would be "BCDEFG".

If the "starting character" specification is past the end of the string value, then an empty sub-string value is selected (e.g., if A has a value of 'XYZ', then A[4,1] will have a value of ''). If the "starting character" specification is negative or zero, then the first character is assumed (e.g., if X has a value of 'JOHN', the X[-5,1] will have a value of 'J').

If the "sub-string length" specification exceeds the remaining number of characters in the string, then the remaining string is selected (e.g., if B has a value of '123ABC', the B[5,10] will have a value of 'BC'). If the "sub-string length" specification is negative or zero, then an empty sub-string is selected (e.g., B[5,-2] and B[5,0] both have a value of '').

Concatenation operations may be performed on strings. Concatenation is specified by a colon (:) or CAT operator. The concatenation of two strings (or sub-strings) is the addition of the characters of the second operand onto the end of the first. For example:

"AN EXAMPLE OF " CAT "CONCATENATION" evaluates to

"AN EXAMPLE OF CONCATENATION"

The precedence of the concatenation operator is higher than any of the arithmetic operators. So if the concatenation operator appears in the same expression with an arithmetic operator, the concatenation operation will be performed first. Multiple concatenation operations are performed from left to right. Parenthesized sub-expressions are evaluated first. The concatenation operator considers both its operands to be string values; for example, the following expression evaluates to "56ABC":

56:"ABC"

NOTE: For the following examples, assume that the current value of A is "ABC123", and the current value of variable Z is "EXAMPLE".

EXPRESSION	EXPLANATION
Z[1,4]	Evaluates to "EXAM".
A : Z[1,1]	Evaluates to "ABC123E".
Z[1,1] CAT A[4,3]	Evaluates to "E123"
5*2:0	2:0 is evaluated first and results in the string "20" (i.e., concatenation operator assumes both operands are strings). 5*"20" is then evaluated and results in 100 (i.e., * operator assumes both operands are numeric. Final result is 100.
A[6,1]+5	Evaluates to 8.
Z CAT A : Z	Evaluates to "EXAMPLEABC123EXAMPLE".
Z CAT " ONE"	Evaluates to "EXAMPLE ONE".

Examples of String Expressions Combined With Arithmetic Examples.

Relational expressions are the result of applying a relational operator to a pair of arithmetic or string expressions.

The relational operators are listed below. A relational operation evaluates to 1 if the relation is true, and evaluates to 0 if the relation is false. Relational operators have lower precedence than all arithmetic and string operators; therefore, relational operators are only evaluated after all arithmetic and string operations have been evaluated.

For purposes of clarification, relational expressions may be divided into two types: arithmetic relations and string relations. An arithmetic relation is a pair of arithmetic expressions separated by any one of relational operators. For example:

```
3 < 4          (3 is less than 4)=(true)=1

3 = 4          (3 is equal to 4)=(false)=0

3 GT 3         (3 is greater than 3)=(false)=0

3 >= 3         (3 is greater than or equal to 3)=(true)=1

5+1 > 4/2      (5 plus 1 is greater than 4 divided by 2)=(true)=1
```

A string relation is a pair of string expressions separated by any one of the relational operators. A string relation may also be a string expression and an arithmetic expression separated by a relational operator (i.e., if a relational operator encounters one numeric operand and one string operand, it treats both operands as strings). To resolve a string relation, character pairs (one from each string) are compared one at a time from leftmost characters to rightmost. If no unequal character pairs are found, the strings are considered to be 'equal'. If an unequal pair of characters are found, the characters are ranked according to their numeric ASCII code equivalents (refer to the LIST OF ASCII CODES in APPENDIX E of this manual). The string contributing the higher numeric ASCII code equivalent is considered to be "greater" than the other string. Consider the following relation:

```
"AAB" > "AAA"
```

This relation evaluates to 1 (true) since the ASCII equivalent of B (66) is greater than the ASCII equivalent of A (65).

If the two strings are not the same length, but the shorter string is otherwise identical to the beginning of the longer string, then the longer string is considered "greater" than the shorter string. The following relation, for example, is true and evaluates to 1:

```
"STRINGS" GT "STRING"
```


OPERATOR SYMBOLS	OPERATION
< or LT	Less than
> or GT	Greater than
<= or LE or =<	Less than or equal to
>= or GE or =>	Greater than or equal to
= or EQ	equal to
# or <> or <> or NE	not equal to
MATCH or MATCHES	pattern matching (see next page)

Relational Operators

EXPRESSION	EXPLANATION
4 < 5	Evaluates to 1 (true).
"D" EQ "A"	Evaluates to 0 (false).
"D" > "A"	ASCII equivalent of D (X'44') is greater than ASCII equivalent of A (X'41'), so expression evaluates to 1.
"Q" LT 5	ASCII equivalent of Q (X'51') is not less than ASCII equivalent of 5 (X'35'), so expression evaluates to 0.
6+5 = 11	Evaluates to 1.
Q EQ 5	Evaluates to 1, if current value of variable Q is 5; evaluates to 0 otherwise.
"ABC" GE "ABB"	Evaluates to 1 (i.e., C is "greater" than B).
"XXX" LE "XX"	Evaluates to 0.

Sample Usage of Relational Expressions.

9.15 MATCHES : RELATIONAL EXPRESSION PATTERN MATCHING

BASIC pattern matching allows the comparison of a string value to a predefined pattern. Pattern matching is specified by the MATCH or MATCHES relational operator.

FORMAT:

```
expression MATCH{ES} "pattern"
```

The MATCH or MATCHES relational operator compares the string value of the expression to the predefined pattern (which is also a string value) and evaluates to 1 (true) or 0 (false). The pattern may consist of any combination of the following:

- An integer number followed by the letter N (which tests for that number of numeric characters).
- An integer number followed by the letter A (which tests for that number of alphabetic characters).
- An integer number followed by the letter X (which tests for that number of any characters).
- A literal string enclosed in quotes (which tests for that literal string of characters).

Consider the following expression:

```
DATA MATCHES "4N"
```

This relation evaluates to 1 if the current string value of variable DATA consists of four numeric characters.

If the integer number used in the pattern is 0, then the relation will evaluate to true only if all the characters in the string conform with the "specification letter" (i.e., N,A, or X). For example:

```
X MATCH "0A"
```

This relation evaluates to 1 if the current string value of variable X consists only of alphabetic characters.

As a further example, consider the following expression:

```
A MATCHES "1A4N"
```

This relation evaluates to 1 if the current string value of variable A consists of an alphabetic character followed by four numeric characters.

EXPRESSION	EXPLANATION
Z MATCHES '9N'	Evaluates to 1 if current string value of variable Z consists of 9 numeric characters; evaluates to 0 otherwise.
Q MATCHES "ON"	Evaluation to 1 if current value of Q is any unsigned integer; evaluates to 0 otherwise.
B MATCH '3N"-"2N"-"4N'	Evaluates to 1 if current value of B is, for example, any social security number; evaluates to 0 otherwise.
B="4N1A2N" C MATCHES B	Evaluates to 1 if current string value of C consists of four numeric characters followed by one alphabetic character followed bytwo numeric characters; evaluates to 0 otherwise.
A MATCHES "ON' . 'ON"	Evaluates to 1 if current value of A is any number containing a decimal point; evaluates to 0 otherwise.
"ABC" MATCHES "#N"	Evaluates to 0.
"XYZ" MATCHES "A"	Evaluates to 1.
"XYZ1" MATCH "4X"	Evaluates to 1.
X MATCHES ' '	Evaluates to 1 if current string value of X is the empty string; evaluates to 0 otherwise.

Sample Usage of Pattern Matching Relation.

9.16 OR - AND : LOGICAL EXPRESSIONS

Logical expressions (also called Boolean expressions) are the result of applying logical (Boolean) operators to relational or arithmetic expressions.

FORMAT:

AND	or	&	Logical And operation
OR	or	!	Logical Or operation

Logical operators operate on the true or false results of relational or arithmetic expressions. (Relational expressions are considered false when equal to zero, and are considered true when equal to one; arithmetic expressions are considered false when equal to zero, and are considered true when not equal to zero.) Logical operators have the lowest precedence and are only evaluated after all other operations have been evaluated. If two or more logical operators appear in an expression, the leftmost is performed first.

Logical operators act on their associated operands as follows:

a OR b is true (evaluates to 1) if a is true or b is true; is false (evaluates to 0) only when a and b are both false.

a AND b is true (evaluates to 1) only if both a and b are true; is false (evaluates to 0) if a is false or b is false or both are false.

consider, for example, the following logical expression:

```
A*2-5>B AND J>J
```

The multiplication operation has highest precedence, so it is evaluated first (i.e., A*2 = result 1). the expression then becomes:

```
result 1 - 5>B AND 7>J
```

The subtraction operation is next (i.e., result 1 - 5=result 2). The expression then becomes:

```
result 2 > B AND 7>J
```

the two relational operators are of equal precedence, so the leftmost is evaluated first (i.e., result 2 > B=result 3, where result 3 has a value of 1 indicating true, or a value of 0 indicating false). the expression then becomes:

```
result 3 AND 7>J
```

The remaining relational operation is then performed (i.e., 7>J = result 4, where result 4 equals 1 or 0). The final expression therefore becomes:

```
result 3 AND result 4
```

which is evaluated as true (1) if both result 3 and result 4 are true, and is evaluated as false (0) otherwise.

The NOT function may be used in logical expressions to negate (invert) the expression or sub-expression (refer to the description of the NOT Intrinsic Function).

EXPRESSION	EXPLANATION
1 AND A	Evaluates to 1 if current value of variable A is non-zero; evaluates to 0 if current value of A is 0.
8-2*4 OR Q5-3	Evaluates to 1 if current value of Q5-3 is non-zero; evaluates to 0 if current value of Q5-3 is 0.
A>5 OR A<0	Evaluates to 1 if the current value of variable A is greater than 5 or is negative; evaluates to 0 otherwise.
1 AND (0 OR 1)	Evaluates to 1.
J EQ 7 AND I EQ 5*2	Evaluates to 1 if the current value of variable J is 7 and the current value of variable I is 10; evaluates to 0 otherwise.
"XYZ1" MATCH "4X" AND X	Evaluates to 1 if the current value of variable X is non-zero; evaluates to 0 if current value of X is 0.
X1 AND X2 AND X3	Evaluates to 1 if the current value of each variable (X1, X2, and X3) is non-zero; evaluates to 0 if the current value of either or all variables is 0.

Sample Usage of Logical Expressions.

Variable values may be formatted via the use of format strings. A format string immediately following a variable name or expression specifies that the value will be formatted as specified by the characters within the format string. The format string may also be used directly in conjunction with the PRINT statement.

FORMAT:

variable = variable"{numeric mask code}{(format mask code)}"

The format string uses the same subroutines as the ACCESS Mask Conversion Code. It may be used to format both numeric and non-numeric strings.

The entire format string is enclosed in quotes. If the format mask is used, it is enclosed in parentheses within the quotes.

The entire format string may be used as a literal, or it may be assigned to a variable. In either case the format string or variable immediately follows the variable it is to format.

The numeric mask code is represented by the symbols: J, n, m, Z, ',', c and $, which controls justification, precision, scaling and credit indication. The format mask code controls field length and fill characters.

The formatted value may be assigned to the same variable or to a new variable (as shown in the general form), or it may be used in a PRINT statement of the form: PRINT X"format string".

The format mask code may be used separately or in conjunction with the numeric mask.

The format mask code is enclosed in parentheses, and may consist any combination of format characters and literal data.

The field length specified ('n') should not exceed 99. The format characters are "#", "*" or "%", optionally followed by a numeric, such as "#3" or "%5".

Any other character in the format field, including parentheses, may be used as a literal character.

NOTE: If a dollar sign is placed outside of the format mask, it will be output just prior to the value, regardless of the filled mask. If a dollar sign is used within the format field it will be output in the leftmost position regardless of the filled field.

NUMERIC MASK CODE:

j specifies justification. "R" specifies right justification. "L" specifies left justification. Default justification is left.

n is a single numeric digit defining the number of digits to print out following the decimal point. If n = 0, the decimal point will not be output following the value.

m is an optional 'scaling factor' specified by a single numeric digit which 'descales' the converted number by the 'mth' power of 10. Because PICK/BASIC assumes 4 decimal places (unless otherwise specified by a Precision Statement) to descale a number by 10 m should be set to 5, to descale a number by 100, m should be set to 6, etc.

Z is an optional parameter specifying the suppression of leading zeros.

, is an optional parameter for output which inserts commas between every thousands position of the value.

c The following five symbols are Credit Indicators which are optional parameters of the form:

- c Causes the letters 'CR' to follow negative values and causes two blanks to follow positive or zero values.
- d Causes the letters 'DB' to follow positive values; two blanks to follow negative or zero values.
- m Causes a minus sign to follow negative values; a blank to follow positive or zero values.
- e Causes negative values to be enclosed with a "<.....>" sequence; a blank follows positive or zero values.
- n Causes the minus sign of negative values to be suppressed.

$ Is an optional parameter for output which appends a dollar

FORMAT MASK CODE:

- #n specifies that the data is to be filled on a field of 'n' blanks.
- *n specifies that the data is to be filled on a field of 'n' asterisks.
- %n specifies that the data is to be filled on a field of 'n' zeros and to force leading zeros into a fixed field. 'D'() specifies the standard system 'D' (date) conversion.

NOTE: Any other character, including parentheses may be used as a field fill.

Explanation of the Format String Codes.

FORMAT:

variable = variable"{numeric mask code}{(format mask code)}"

NUMERIC MASK

MASK CODE	IMPLEMENTED AS	MEANING
j	R or L	Right or Left justification (default is left justification).
n	single numeric	# of decimal places.
m	single numeric	'Descaling' factor.
Z	Z	Suppress leading zeros.
,	,	Insert commas every thousands position.
c	C,D,M or E	Credit indicators.
$	$	Outputs dollar sign prior to value.

FORMAT MASK (enclosed in parentheses)

MASK CODE	IMPLEMENTED AS	MEANING
$	$	Outputs a dollar sign in the leftmost position of field.
#n	#10	Fills data on a field of 10 blanks.
%n	%10	Fills data on a field of 10 zeros.
*n	*10	Fills data on a field of 10 asterisks, or on a field of any other specified character.

NOTE: If a dollar sign is placed outside of the format mask, it will be output just prior to the value, regardless of the filled field. If a dollar sign is used within the format mask it will be output in the leftmost position regardless of the filled field.

General Form and Summary of Format String Codes.

UNCONVERTED STRING (X)	FORMAT STRING	RESULT (V)
X = 1000	V = X"R26"	10.00
X = 1234567	V = X"R27,"	1,234.57
X = -1234567	V = X"R27,E$"	$<1234.57>
X = 38.16	V = X"1"	38.2
X = -1234	V = X"R25$,M(*10#)"	***$123.40-
X = -1234	V = X"R25,M($*10#)"	$****123.40-
X = -1234	V = X"R25,M($*10)"	$***123.40-
X = 072458699	V = X"L(###-##-####)"	072-45-5866
X = 072458699	V = X"L(#3-#2-#4)"	072-45-5866
X = SMITH, JOHANNSEN	V = X"L((#13))"	(SMITH, JOHANN)

Sample usage of Numeric & Format Codes.

9.18 @ FUNCTION : CURSOR CONTROL

The @ function generates the terminal output codes required to position the cursor to a specified position.

FORMAT:

@(column{,row})

The column parameter specifies column cursor address at which to position the cursor. (at the current line if row not specified). The row parameter generates the cursor row address at which to position the cursor.

The values of the expression(s) used in the @ function must be within the row and column limits of the terminal screen.

When the @ function is used with a negative value, it generates the special cursor-control characters for the current terminal type (as defined by the TERM statement in effect at the time).

@(-1)	Clear screen and 'home' cursor. (upper-left corner)
@(-2)	Positions the cursor at 'home' (upper left corner).
@(-3)	Clears from cursor positon to the end of the screen.
@(-4)	Clears from cursor position to the end of the line.
@(-5)	Starts blinking on subsequently printed data.
@(-6)	sStops blinking. @(-6)
@(-7)	Initiates 'protect' field.
@(-8)	Stops protect field. @(-8)
@(-9)	Backspaces the cursor one character.
@(-10)	Moves the cursor up one line. @(-10)

CURSOR FUCTION VALUES.

STATEMENT	EXPLANATION
X = 7 ; Y = 3 PRINT @(X,Y) : Z	Prints the current value of variable Z at column position 7 of line 3.
Q = @(3 "HI" PRINT Q	Prints "HI" at column position 3 of current line.
A = 5 PRINT @(A,A+5):A	Prints the value 5 at column position 5 of line 10.
PRINT @(-1)	Clears the screen and positions the cursor at 'home' position.

Sample usage of the @ Function.

9.19 ABORT STATEMENT : TERMINATION

The ABORT statement may appear anywhere in the program; it designates a logical termination of the program.

FORMAT:

```
ABORT {errnum{,param, param, ...}}
```

Upon the execution of a ABORT statement, the PICK/BASIC program will terminate.

The ABORT statement may be placed anywhere within the PICK/BASIC program to indicate the end of one of several alternative paths of logic. The ABORT statement is similiar to the STOP statement except that the ABORT statement will terminate execution of any PROC which might have called the program containing the ABORT statement.

Like the STOP statement, the ABORT statement may optionally be followed by an error message name, and error message parameters separated by commas. The error message name is a reference to an item in the ERRMSG file. The parameters are variables or literals to be used within the error message format.

```
PRINT 'PLEASE ENTER FILE NAME':
INPUT FN
OPEN '', FN TO FFN ELSE ABORT 201, FN
```

This program requests a file name and attempts to open the file. If an incorrect file name is entered, the standard system error message 201 "xxx IS NOT A FILE" will be printed, and the program is terminated.

Sample usage of the ABORT statement.

9.20 ABS FUNCTION : ABSOLUTE NUMERIC VALUE

The ABS function returns an absolute value. An absolute value is an unsigned integer value.

FORMAT:

ABS(expression)

The ABS function generates the absolute numeric value of the expression.

An absolute value is the numerical value of a number without reference to its algebraic sign. The result looks positive, but it is in fact, unsigned. For example:

```
A = 100 ; B = 25
C = ABS(B-A)
```

These statements assign the value 75 to variable C. (An absolute value is conventionally written as |75| .)

STATEMENT	EXPLANATION
A = ABS(Q)	Assigns the absolute value of variable Q to variable A.
A = 600 B = ABS(A-1000)	Assigns the value 400 to variable B.
A = 3 B = -10 C = ABS(A+B)	Assigns the value 7 to variable C.

Sample Usage of the ABS Function.

The ALPHA function returns a value of true (1) if the given expression evaluates to a alphabetic character or string.

FORMAT:

ALPHA(expression)

The ALPHA function tests the given expression for a alphabetic value. For example, if the expression evaluates to a alphabetic character or alphabetic string the ALPHA function will return a value of true (i.e., generating a value of 1).

Inversely, an expression evaluating to a number or an numeric string will cause the ALPHA function to return a value of false. Consider the following examples:

```
IF ALPHA(expression) THEN PRINT "ALPHABETIC DATA"
```

This statement will print the text "ALPHABETIC DATA" if the current value of variable "expression" is a letter or an alphabetic string.

In the case of a non-numeric, non-alphabetic character or string (#, ?, ., etc.) a value of false would be returned for both the ALPHA and NUM functions.

The empty string ('') is considered to be a numeric string, but not an alpha string.

(See: NUM)

STATEMENT	EXPLANATION
A1=ALPHA("ABC")	Assigns a value of 1 to variable A1.
A3=ALPHA("12C")	Assigns a value of 0 to variable A3.
IF ALPHA(I CAT J) THEN GOTO 5	Transfers control to statement 5 if current value of both variables I and J are letters or alphabetic strings.

Sample Usage of NOT, NUM and ALPHA Functions.

The ASCII function converts a string value from EBCDIC to ASCII.

FORMAT:

ASCII(expression)

The string value of the expression is converted from EBCDIC to ASCII. For example:

```
A = ASCII(B)
```

Conversely, the EBCDIC function is available to convert string values from ASCII to EBCDIC.

(See: EBCDIC)

STATEMENT	EXPLANATION
READT X ELSE STOP Y = ASCII(X)	Reads a record from the magnetic tape unit and assigns value to variable X. Assigns ASCII value of record to variable Y.

Sample Usage of the ASCII function.

9.23 ASSIGNMENT STATEMENT : ASSIGNING VARIABLE VALUES

The Simple Assignment statement is used to assign a value to a variable.

FORMAT:

```
variable = expression
```

The resultant value of the expression becomes the current value of the variable on the left side of the equality sign. The expression may be any legal PICK/BASIC expression. For example:

```
ABC = 500
X2 = (ABC+100)/2
```

The first statement will assign the value of 500 to the variable ABC. The second statement will asign the value 300 to the variable X2 (i.e., X2 = (ABC+100)/2 = (500+100)/2 = 600/2 = 300).

String values may also be assigned. For example:

```
VALUE = "THIS IS A STRING"
SUB = VALUE [6,2]
```

The first statement above assigns the string "THIS IS A STRING" to variable VALUE. The second statement assigns the string "IS" to variable SUB (i.e., assigns to SUB the 2 character sub-string starting at character position 6 of VALUE).

STATEMENT	EXPLANATION
X=5	Assigns 5 to X.
X=X+1	Increments X by 1.
ST="STRING"	Assigns the character string to ST.
ST1=ST[3,1]	Assigns sub-string "R" to ST1.
TABLE(I,J)=A(3)	Assigns matrix element from vector element.
A=B=0	Assigns 1 to A if "B=0" is true, assigns 0 to A if "B=0" is false.

Examples of Assignment Statements.

9.24 BREAK ON AND OFF : DEBUGGER INHIBITION

The BREAK statements enable or disable the Debugger function accordingly.

FORMAT:

```
BREAK ON
BREAK OFF
BREAK expression
```

These commands increment/decrement the break inhibit counter. Note that they are cummulative. If two BREAK OFFs are executed, two BREAK ONs must be executed to restore a breakable status.

If the expression form of the command is used, the break key is disabled when the expression evaluates to 0. The break key is enabled when the expression evaluates to non-zero.

(See: PICK/BASIC DEBUGGER)

The CALL and SUBROUTINE statements provide external subroutine capabilities for the PICK/BASIC program. An external subroutine is a subroutine that is compiled and cataloged separately from the program or programs that call it.

FORMAT:

```
CALL name (argument list)
SUBROUTINE name (argument list)
```

The CALL statement transfers control to the cataloged subroutine 'name'. The CALL 'argument list' consists of zero or more expressions, separated by commas, that represent actual values passed to the subroutine. The SUBROUTINE 'argument list' consists of the same number of expressions, by which the subroutine references the values being passed to it.

The SUBROUTINE statement is used to identify the program as a subroutine and must be the first statement in the program.

There is no correspondence between variable names or labels in the calling program and the subroutine. The only information passed between the calling program and the subroutine are the arguments. A sample external subroutine that involves two arguments together with correctly formed CALL statements, is shown below.

```
CALL Statements              Subroutine ADD
CALL ADD (A,B,C)             SUBROUTINE ADD (X,Y,Z)
CALL ADD (A+2,F,X)           Z=X+Y
CALL ADD (3,495,Z)           RETURN
                             END
```

An external subroutine must contain a SUBROUTINE statement, a RETURN statement, and an END statement. GOSUB and RETURN may be used in the subroutine. When a RETURN is executed with no corresponding GOSUB, control passes to the statement following the corresponding CALL statement. If the subroutine's END statement, a STOP or CHAIN statement (see appropriate section of the manual) is executed, control never returns to the calling program. The CHAIN statement should not be used to chain from an external subroutine to another PICK/BASIC program.

STATEMENTS	EXPLANATION
CALL REVERSE (A,B) SUBROUTINE REVERSE (I,X)	Subroutine REVERSE has two arguments.
CALL REPORT SUBROUTINE REPORT	Subroutine REPORT has no parameters.
CALL DISPLAY (A,B, C) SUBROUTINE DISPLAY (I,J,K)	Subroutine DISPLAY accepts (and returns) three argument values.

Sample Usage of CALL and SUBROUTINE Statements.

9.26 ARRAY PASSING AND THE CALL @ STATEMENT : INDIRECT EXTERNAL SUBROUTINES

Arrays may be passed to external subroutines. External subroutines may be called indirectly.

PASSING ARRAYS TO EXTERNAL SUBROUTINES PASSING ARRAYS TO EXTERNAL SUBROUTINES

FORMAT:

```
MAT variable
```

The 'variable' is the name of the array given in the DIMENSION statement. The array must be dimensioned in both the calling program and the subroutine. Array dimensions may be different, as long as the total number of elements matches. Arrays are copied in row major order. Consider the following example:

```
Calling Program                 Subroutine
DIM X(10), Y(10)                SUBROUTINE COPY (MAT A)
CALL COPY (MAT X, MAT Y)        DIM A(10,2)
END                             PRINT A(15)
                                RETURN
                                END
```

In this subroutine the parameter passing facility is used to copy MAT X and MAT Y specified in the CALL statement of the calling program into MAT A of the subroutine. Printing A(15) in the subroutine is equivalent to printing Y(5) in the calling program.

INDIRECT FORM OF THE CALL STATEMENT

FORMAT:

```
CALL @name (argument list)
```

The 'name' is a variable containing the name of the cataloged subroutine to be called. The argument list performs the same function as in a direct call.

```
NAME = 'XSUB1'
CALL @NAME
NAME = 'XSUB2'
CALL @NAME
```

The first call invokes subroutine XSUB1. The second call invokes subroutine XSUB2.

STATEMENTS	EXPLANATION
DIM A(4,10),B(10,5) CALL REV (MAT A, MAT B)	Subroutine REV accepts two input array variables, one of size 40 and one of size 50 elements.
SUBROUTINE REV (MAT C, MAT B) DIM C(4,10), B(50)	

Examples of Array Parameters.

9.27 CASE STATEMENT : CONDITIONAL BRANCHING

The CASE statement provides conditional selection of a sequence of BASIC statements.

FORMAT:
```
BEGIN CASE
    CASE expression
    statements
    CASE expression
    statements
    . . .
END CASE
```

If the logical value of the first expression is true (i.e., non-zero), then the statement or sequence of statements that immediately follows is executed, and control passes to the next sequential statement following the entire CASE statement sequence. If the first expression is false (i.e., zero), then control passes to the next test expression, and so on. Consider the following example:

```
BEGIN CASE
    CASE A < 5
    PRINT 'A IS LESS THAN 5'
    CASE A < 10
    PRINT 'A IS GREATER THAN OR EQUAL TO 5 AND LESS THAN 10'
    CASE 1
    PRINT 'A IS GREATER THAN OR EQUAL TO 10'
END CASE
```

If A<5, then the first PRINT statement will be executed. If 5<=A<10, then the second PRINT statement will be executed. Otherwise, the third PRINT statement will be executed. (Note that a test expression of 1 means "always true.")

STATEMENTS	EXPLANATION
BEGIN CASE CASE Y=B Y=Y+1 END CASE	Increment Y if Y is equal to B. Note that this single-case example is equivalent to the statement IF Y=B THEN Y=Y+1.
BEGIN CASE CASE A=0; GOTO 10 CASE A<0; GOTO 20 CASE 1; GOTO 30 END CASE	Program control branches to the statement with label 10 if the value of A is zero; to 20 if A is negative; or to 30 if A is greater than zero.
BEGIN CASE CASE ST MATCHES "1A" MAT LET=1 CASE ST MATCHES "1N" SGL=1; A.1(I)=ST CASE ST MATCHES "2N" DBL=1; A.2(J)=ST CASE ST MATCHES "3N" GOSUB 103 END CASE	If ST is one letter, "1" is assigned to all LET elements and the entire CASE is ended. If ST is one number, "1" is assigned to SGL, ST is stored at element A.1(I), and the entire case is ended. If ST is two numbers, "1" is assigned to DBL, ST is stored at element A.2(J), and the entire case is ended. If ST is three numbers, subroutine 103 is executed.

Sample usage of the CASE statement.

The CHAIN statement allows a PICK/BASIC program to execute any valid TCL command, including the ability to pass values to a separately compiled PICK/BASIC program which is executed during the same terminal session.

FORMAT:

CHAIN "any tcl command"

The CHAIN statement causes the specified TCL command to be executed. The CHAIN statement may contain any valid Verb or PROC name in the user's Master Dictionary. Consider the following example:

CHAIN "RUN FILE1 PROGRAM1 (I)"

This statement causes the previously compiled program named PROGRAM1 in the file named FILE1 to be executed. The I option specifies that the data area is not to be re-initialized.

The CHAIN statement allows values to be passed to the specifed program. This is possible since all PICK/BASIC programs which are executed during a single terminal session use the same data area. The variable in one program that are to be passed to another program must be in the same location. This is accomplished via use of the DIM statement. Consider, for example, the following two PICK/BASIC programs:

Program ABC in file BP

```
DIM A(1,1), B(2)
A=500
B(1)=1 B(2)=2
CHAIN "RUN BP XYZ (I)"
END
```

Program XYZ in file BP

```
DIM I(2), J(1,1)
PRINT I,J
END
```

Program ABC causes program XYZ to be executed. The I option used in the CHAIN statement (refer to the section titled USAGE PROCEDURES) specifies that the data area is not to be re-initialized, thus allowing program ABC to pass the values "500", "1", and "2" to program XYZ. Program XYZ, in turn, prints the values "500", "1", and "2". All dimensioned variables form a long vector in row major order, and on a the chain are assigned left to right to chained program's dimensioned variables.

The user should note that control is never returned to the PICK/BASIC program originally executing the CHAIN statement.

STATEMENT	EXPLANATION
CHAIN "RUN FN1 LAX (I)"	Causes the execution of program LAX in file FN1. I option specifies that data area is not to be re-initialized (i.e., the program executing the CHAIN statement will pass values to program LAX).
CHAIN "LISTU"	Causes the execution of the LISTU SYSPROG PROC.
CHAIN "LIST FILE"	Causes the execution of the LIST ACCESS Verb.
CHAIN "RUN PROGRAMS ABC"	Causes the execution of program ABC in file PROGRAMS. Since I option is not used, values will not be passed to program ABC.

Sample usage of the CHAIN statement.

9.29 CHAR FUNCTION : FORMAT CONVERSION

The CHAR function converts a numeric value to its corresponding ASCII character.

FORMAT:

CHAR(expression)

The CHAR function converts the numeric value specified by the expression to its corresponding ASCII character string value. For example, the following statement assigns the string value for as Attribute Mark to the variable AM:

```
AM = CHAR(254)
```

Conversely, the SEQ function is available to convert the first character of a string value to its corresponding numeric decimal value.

NOTE: For a complete list of ASCII codes, refer to the Appendix.

STATEMENT	EXPLANATION
SM = CHAR(255)	Assigns the string value for a Segment Mark to variable SM.
X = 252 SVM = CHAR(X)	Assigns the string value for a Secondary Value Mark to variable SVM.

Sample Usage of the CHAR Function.

9.30 CLEAR STATEMENT : INITIALIZING VARIABLE VALUES

The CLEAR statement is used to initialize all variables to a value of zero.

FORMAT:

```
CLEAR
```

The CLEAR statement initializes all possible variables to zero (i.e., assigns the value 0 to all variables). The CLEAR statement may be used in the beginning of the program to initialize all variables to zero, or may be used anywhere within the program for re-initialization purposes.

STATEMENT	EXPLANATION
CLEAR	Assigns the value 0 to all possible variables.

Example of the CLEAR Statement.

The CLEARFILE statement is used to clear out the data section of a specified file.

FORMAT:

CLEARFILE {file.variable}

Upon execution of the CLEARFILE statement, the data section of the file which was previously assigned to the specified file.variable via an OPEN statement, will be emptied. The data in the file will be deleted, but the file itself will not be deleted. If the file.variable is omitted from the CLEARFILE statement, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

The dictionary section of file cannot be cleared via a CLEARFILE statement. A PICK/BASIC program will abort with an appropriate error message if the specified file has not been opened prior to the execution of the CLEARFILE statement.

STATEMENT	EXPLANATION
OPEN 'FN1' ELSE PRINT 'NO FN1';STOP READ I FROM 'I1' ELSE STOP CLEARFILE	Opens the data section of file FN1, reads item I1 and assigns value to variable I, and finally clears the data section of file FN1.
OPEN 'FILEA' TO A ELSE STOP OPEN 'FILEB' TO B ELSE STOP CLEARFILE A CLEARFILE B	Clears the data sections of files FILEA AND FILEB.
OPEN 'ABC' ELSE PRINT 'NO FILE'; STOP READV Q FROM 'IB3',5 ELSE STOP IF Q = 'TEST' THEN CLEARFILE	Clears the data section of file ABC if the 5th attribute of the item with name IB3 has a string value of 'TEST'.

Sample usage of the CLEARFILE statement.

The COL1 and COL2 functions return the numeric values of the column positions immediately preceding and immediately following the sub-string selected by the FIELD function.

FORMAT:

```
COL1()
COL2()
```

COL1() returns the numeric value of the column position immediately preceding the sub-string selected via the most recent FIELD function. For example:

```
B = FIELD("XXX.YYY.ZZZ.555",".",2)
BEFORE = COL1()
```

These statements assign the numeric value 4 to the variable BEFORE (i.e., the value "YYY" which is returned by the FIELD function is preceded in the original string by column position 4).

COL2() returns the numeric value of the column position immediately following the sub-string selected via the most recent FIELD function. COL2() returns zero if the sub-string is not found. For example:

```
B = FIELD("XXX.YYY.ZZZ.555",".",2)
AFTER = COL2()
```

These statements assign the numeric value 8 to the variable AFTER (i.e., the value "YYY" which is returned by the FIELD function is followed in the original string by column position 8).

(See: FIELD)

STATEMENT	EXPLANATION
Q = FIELD("ABCBA","B",2) R = COL1() S = COL2()	Assigns the string value "C" to variable Q, the numeric value 2 to variable R, and the numeric value 4 to variable S.

Sample Usage of the COL1() and COL2() Functions.

9.33 COMMON STATEMENT : VARIABLE SPACE ALLOCATION

The COMMON statement may be used to control the order in which space is allocated for the storage of variables, and for the passing of values between programs.

FORMAT:

COM{MON} variable {,variable}...

The purpose of the COMMON statement is to change the automatic allocation sequence that the compiler follows, so that more than one program may have specified variables in a pre-determined sequence.

In the absence of a COMMON statement, variables are allocated space in the order in which they appear in the program, with the additional restriction that arrays are allocated space after all simple variables. Common variables (including Common arrays) are allocated space before any other variables in the program. The COMMON statement must appear before any of the variables in the program are used.

The COMMON variable list may include simple variables, file variables and arrays. Arrays may be declared in a COMMON statement by specifiying the dimensions enclosed in parentheses, (e.g. COMMON A(10) declares an array "A" with 10 elements). Arrays that are declared in a COMMON statement should not be declared again by a DIMENSION statement. All variables in the program which do not appear in a COMMON statement are allocated space in the normal manner.

The COMMON statement may be used to share variables among CHAINed programs, or among main-line programs and subroutines. This ensures that all 'COMMON' variables refer to the same stored values in different programs. (The 'I' option must be used with the RUN verb in chained programs to inhibit re-initialization.) For example:

```
COMMON X,Y,Z(5)
COMMON Q,R,S(5)
```

If the first statement is found in a main-line program and the second in a subroutine call it is ensured that the variables X and Q, Y and R, and the arrays Z and S share the same locations. NOTE: The second COMMON statement variables may be regarded as a mask over the first. What associates Q to X (R to Y and S to Z) is a matter of alignment. Thus if the second statement had been "COMMON Q(2),R(5)" then Q(1) would refer to the location where the value of X is stored and Q(2) would refer to the location where the value of Y is stored.

The COMMON statement differs from the argument list in a Subroutine Call in that the actual storage locations of Common variables are shared by the main-line program and its external subroutines; whereas the argument list in a Subroutine Call causes the values to be pushed on to the stack. The COMMON statement thereby affords a more efficient method of passing values.


```
 Item "MAINPROG"

COMMON A,B,C(10)            Variables A, B, and array C are
A = "NUMBER"                allocated space before any other
B = "SQUARE ROOT"           variables.
FOR I = 1 TO 10
 C(I) = SQRT(I)
NEXT I
CALL SUB                    Subroutine call to program SUBPROG.
PRINT "DONE"
END

 Item SUBPROG

COMMON X(2),Y(10)           The 2 elements of array X contain
PRINT X(1), X(2)            respectively, the values of A and
FOR J = 1 TO 10             B from the main-line program.  The
 PRINT J, Y(J)              array Y contains the values of C
NEXT J                      from the main-line program.
RETURN                      Returns to main-line program.
END
```

Sample usage of the COMMON statement.

9.34 COS FUNCTION : COSINE OF AN ANGLE

The COS function generates the trigonmetric cosine of an angle.

FORMAT:

COS(expression)

To generate the cosine of an angle expressed in degrees the COSINE function is used. The given angle must be less than or equal to 14,073,748,834 and greater than or equal to -14,073,748,834.

Values which are less than 0 degrees, or greater than 360 degrees are adjusted to this range before generation.

(See: SINE)

STATEMENT	EXPLANATION
YY = COS(XX)	Assigns the cosine of an angle of XX degrees to YY.
PRINT COS(1)	Prints "0.9998"
PRINT COS(361)	Prints "0.9998"
PRINT COS(2)	Prints "0.9994"
PRINT COS(362)	Prints "0.9994"
PRINT COS(45)	Prints "0.7071"
PRINT COS(90)	Prints "0"

Sample usage of the COS function.

The COUNT function counts the number of occurances of a substring within a string.

FORMAT:

COUNT(string,substring)

The COUNT function counts the number of occurances of a substring within a string. Any number of characters may be present in the substring. This function is particularly useful for determing the number of attributes within an item, or the number of multiple values or sub-values within an attribute.

The COUNT function returns a value of zero if the substring is not found, and returns the number of characters in the string if the substring is null (i.e. a null matches on any character). For example:

```
    COMMAND                                    VALUE OF X

    X = COUNT('THIS IS A TEST','IS')             2
    X = COUNT('THIS IS A TEST','X')              0
    X = COUNT('THIS IS A TEST','')              14
```

(There are 14 characters in the string.)

```
    X = COUNT('AAAA','AA')                       3
```

There are 3 substrings within the string AAAA.

```
    AAAA              STRING
    XX                SUBSTRING 1
     XX               SUBSTRING 2
      XX              SUBSTRING 3
```

(See: DCOUNT)

STATEMENT	EXPLANATION
A = "1234ABC5723" X = COUNT(A,'23')	Value returned in X is 2 as there are two occurances of '23' in the string A.
X = COUNT('ABCDEFG','')	Value returned in X is 7 as a null substring will match any character.

Sample examples of the COUNT function.

9.36 DATA STATEMENT : STACKING INPUT DATA

The DATA statement is used to store data for stacked input when using the CHAIN statement.

FORMAT:

```
DATA expression{,expression ...}
```

Where 'expression' may be any valid combination of variables, literals, functions, etc. Each expression becomes the response to one input request from the CHAINed process.

Each DATA statement will generate one line of stacked input. The lines of stacked input are then used in response to the input requests of other processes. The DATA statement may be used to store stacked input for ACCESS, TCL, PROCs, or other PICK/BASIC programs.

The following example illustrates the procedure to exit a PICK/BASIC program, sort-select a file and begin execution of a second PICK/BASIC program. The variable REF-DATE is passed to the second PICK/BASIC program. Assuming that no stacked input is currently present:

```
DATA 'RUN BP PROG';  DATA REF-DATE
CHAIN 'SSELECT FILE WITH DATE  "':REF.DATE:'" BY DATE'
```

The first statement stacks two values (e.g. 'RUN BP PROG' and 'REF-DATE'). The second statement causes an ACCESS statement to be executed. When the ACCESS processor has completed, the first value on the stack is the input to the TCL prompt, thus BP PROG begins execution. (Note that the stack is a First In First Out (FIFO) type.)

NOTE: the DATA statement must be processed before the CHAIN statement!! The second PICK/BASIC program (BP PROG) then performs the following:

```
INPUT REF-DATE
```

This instruction gets its input from the second value on the stack, i.e. the value of REF-DATE from the first PICK/BASIC program.

STATEMENT	EXPLANATION
DATA A DATA B DATA C CHAIN 'RUN BP TEST'	Stacks the values of A, B and C for subsequent input requests. Program 'TEST' may have three input requests which will be satisfied by the stacked input.
DATA 'RUN BP CHARGE-ACC' DATA DATE CHAIN 'SELECT ACC WITH AMT > 100'	This causes the TCL command 'RUN BP CHARGE-ACC' to be stored on the stack. Control first exits to the ACCESS processor to perform the SELECT, after which the PICK/BASIC program is run with DATE as stacked input.

Sample usage of the DATA statement.

9.37 DATE() FUNCTION : DATE CAPABILITY

The DATE() function returns the current internal date.

FORMAT:

DATE()

The DATE() function returns the string value containing the internal date. The internal date is the number of days since December 31, 1967.

(See: TIME() and TIMEDATE() functions)

STATEMENT	EXPLANATION
Q = DATE()	Assigns string value of current internal date to variable Q.
PRINT DATE()	Prints the current date in the internal format.
WRITET DATE() ELSE STOP	Writes the string value of the current internal date onto a magnetic tape record.

Sample Usage of the DATE() function.

The DCOUNT Function returns a value which is the number of values separated by a specified delimiter.

FORMAT:

DCOUNT(string,substring)

The DCOUNT function counts the number of values separated by a specified delimiter. The DCOUNT function differs from the COUNT function in that it returns the true number of values by the specified delimiter, rather than the number of occurances of the delimiter within the string. For example, considering the string:

A = ABC^DEF^GHI^JKL

COMMAND	VALUE OF X
X = COUNT(A,AM)	3
X = DCOUNT(A,AM)	4

The DCOUNT function may be used to count the number of attributes in an item, or the number of values (or subvalues) within an attribute. The DCOUNT function returns a value of zero when a null string is encountered.

(See: COUNT)

STATEMENT	EXPLANATION
AM = CHAR(254) A = "123^456^ABC" X = DCOUNT(A,AM)	Value returned in X is 3 as there are three values in the string separated by attribute marks.
VM = CHAR(253) A = "123]456^ABC]DEF]HIJ" X = DCOUNT(A,VM)	Value returned in X is 4 as there are four values in the string separated by value marks.
A = "ABCDEFG" X = DCOUNT(A,'')	Value returned in X is 0 as a null is specified as the delimiter.

Examples of DCOUNT function.

9.39 DELETE STATEMENT : DELETING ITEMS

The DELETE statement is used to delete a file item.

FORMAT:

DELETE {file.variable,} itemname

The DELETE statement deletes the item which is specified by the itemname and which is located in the file previously assigned to the specified file.variable via an OPEN statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

No action is taken if a non-existent item is specified in the DELETE statement.

The user should note that the PICK/BASIC program will abort with an appropriate error message if the specified file has not been opened prior to the execution of the DELETE statement.

(See: DELETE Function)

STATEMENT	EXPLANATION
DELETE X, "XYZ"	Deletes item XYZ in the file opened and assigned to variable X.
Q="JOB" DELETE Q	Deletes item JOB in the file opened without a file variable.

Sample Usage of the DELETE Statement.

9.40 DELETE FUNCTION : DYNAMIC ARRAY DELETION

The DELETE function deletes an attribute, a value, or a secondary value from a string in 'item' format (called a dynamic array).

FORMAT:

DELETE(da.expression,att#{,value#,sub-value#})

The dynamic array used by this function is specified by the da.expression. Whether an attribute, a value, or a secondary value is deleted depends upon the values of the second, third, and fourth parameters. The att# specifies an attribute, the value# specifies a value, and the sub-value# specifies a secondary value. If the value# and sub-value# both have a value of 0, or are dropped, then an entire attribute is deleted. If the last three expressions are all non-zero, then a secondary value is deleted.

If a value is deleted the value mark associated with the value is also deleted. If an attribute is deleted the attribute mark associated with the attribute is also deleted. Consider the following example:

```
OPEN 'TEST' TO TEST ELSE STOP 201,'TEST'
READ X FROM TEST,'NAME' ELSE STOP 202,'NAME'
WRITE DELETE(X,2) ON TEST,'NAME'
```

These statements delete attribute 2 (and its associated delimiter) of item NAME in file TEST.

STATEMENT	EXPLANATION
Y = DELETE(X,3,2)	Deletes value 2 of attribute 3 of dynamic array X (and its associated delimiter), and assigns resultant dynamic array to Y.
A=1;B=2;C-3 DA = DELETE(DA,A,B,C-A)	D eletes secondary value 2 (and its associated delimiter) of value 2 of attribute 1 of dynamic array DA.
X = DELETE (X,7)	Deletes attribute 7 (and its associated delimiter) of dynamic array X.
PRINT DELETE(X,7,1)	Prints the dynamic array which results when value 1 of attribute 7 of dynamic array X is deleted.

Sample usage of the DELETE Function.

Multiple valued variables are called arrays. Before arrays may be used in a PICK/BASIC program they must be dimensioned via a DIM statement.

FORMAT:

DIM variable (dimension1{,dimension2})

A variable with more than one value associated with it is called an array. Each value is called an element of the array, and the elements are ordered. Before an array may be used in a PICK/BASIC program, however, the maximum dimension(s) of the array must be specified for storage purposes. This is done via a DIM statement, wherein the dimensions of an array are declared with constant whole number, separated by commas. DIM statements must precede any array references, and are therfore usually placed at the beginning of the program. (Arrays need only be dimensioned once throughout the entire program.) Several arrays may be dimensioned via a single DIM statement.

```
          -------
          |  3  |---- The first element of A has value 3
          -------
          |  8  |---- The second element of A has value 8
Array A:  -------
          |-20.3|---- The third element of A has value -20.3
          -------
          | ABC |---- The fourth element of A has string value "ABC"
          -------
```

The above example illustrates a one-dimensional array (called a vector). A two-dimensional array (called a matrix) is characterized by having rows and columns. For example:

```
                COL.1  COL.2  COL.3  COL.4
               ------------------------------
          Row 1 |  3  | XYZ |   A  | -8.2 |
               |-------------------------|
Array Z:  Row 2 |  8  | 3.1 |  500 | .333 |
               |-------------------------|
          Row 3 |  2  |  -5 | Q123 |  84  |
               ------------------------------
```

Any array element may be accessed by specifying its position in the array. This position is like an offset from the beginning of the array. In specifying an element, the user must have one offset or subscript for each dimension of the array. In Array A, element A(1) has a value of 3, while element A(3) has a value of "20.3". For a two-dimensional array (matrix) the first subscript specifies the row, while the second specifies the column. For example, in array Z above, element Z(1,1) has a value of 3, while element Z(2,3) has a value of 500.

DIM MATRIX(10,12)	Specifies 10 by 12 matrix named MATRIX.
DIM Q(10),R(10),S(10)	Specifies three vectors named Q, R, and S (each to contain 10 elements).
DIM M1(50,10),X(2)	Specifies 50 by 10 matrix named M1, and two-element vector named X.

Sample usage of the DIM statement.

9.42 EBCDIC FUNCTION : FORMAT CONVERSION

The EBCDIC function converts a string value from ASCII to EBCDIC.

FORMAT:

EBCDIC(expression)

The string value of the expression is converted from ASCII to EBCDIC. For example:

```
B = EBCDIC(A)
```

Conversely, the ASCII function is available to convert string values from EBCDIC to ASCII.

(See: ASCII)

STATEMENT	EXPLANATION
B = EBCDIC(A)	Assigns the EBCDIC value of variable A to variable B.

Sample Usage of the EBCDIC function.

9.43 ECHO ON AND OFF : TERMINAL DISPLAY

The ECHO statement enables or disables terminal output accordingly.

FORMAT:

```
ECHO ON
ECHO OFF
ECHO expression
```

These commands turn the system echo-back on or off. They may be used to suppress the echo back of terminal input.

If the expression form is used, terminal echo is inhibited when the expression evaluates to zero. Terminal echo is enabled when the expression evaluates to non-zero.

9.44 END STATEMENT

The END statement must be the last statement of the PICK/BASIC program; it designates the physical end of the program. The STOP and ABORT statement may appear anywhere in the program; it designates a logical termination of the program.

FORMAT:

```
END
```

The END statement may appear as the very last statement in the BASIC program. It is used to specify the physical end of the sequence of statements comprising the program, and increases readability.

The END statement is also used to designate the physical end of alternative sequences of statements within the IF statement and within certain of the PICK/BASIC I/O Statements.

(See: IF..THEN, LOCATE, LOCK, READ for a discussion of this alternative use of the END statement.)

```
      A=500 ; B=750 ; C=235 ; D=1300
      REVENUE = A + B ; COST = C + D
      PROFIT = REVENUE - COST
      IF PROFIT > 1 THEN GOTO 10
      PRINT "ZERO PROFIT OR LOSS"
      STOP
 10   PRINT "POSITIVE PROFIT"
      END <---------------------------- Physical end of program
```

Sample usage of the END Statement.

The ENTER statement permits transfer of control from one cataloged program to another cataloged program. The program that executes the ENTER statement must be executed via the cataloged verb in the user's MD.

FORMATS:

```
ENTER program-name
```

where program-name is the item-id of the program to be ENTERed and

```
ENTER @variable
```

where variable has been assigned the program name to be ENTERed.
All variables which are to be passed between programs must be declared in a COMMON declaration in all program segments that are to be ENTERed.

All other variables will be initialized upon ENTERing the program. It is permissible to ENTER a program that calls a subroutine, but it is illegal to ENTER a program from a subroutine.

STATEMENT	EXPLANATION
ENTER PROGRAM.1	Causes execution of the cataloged program "PROGRAM.1". Any COMMON variables will be passed to "PROGRAM.1".
N=2 PROG = "PROGRAM." : N ENTER @PROG	Causes execution of the cataloged program "PROGRAM.2". Any COMMON variables will be passed to "PROGRAM.2".

Sample usage of the ENTER statement.

9.46 EQUATE STATEMENT : VARIABLE ASSIGNMENT

The EQUATE statement allows one variable to be defined as the equivalent of another variable.

FORMAT:

EQU{ATE} variable TO equate-variable{,variable TO equate-variable..}

The variable must be a simple variable. The equate-variable may be a literal number, string, character or array element. The equate-variable may also be a CHAR function, however, the CHAR function is the only allowed function in an EQUATE statement. The EQUATE statement must appear before the first reference to the equate-variable.

The EQUATE Statement differs from the ASSIGNMENT Statement (where a variable is assigned a value via an equal sign) in that there is no storage location generated for the variable. The advantage this offers is that the value is compiled directly into the object-code item at compile time and does not need to be re-assigned every time the program is executed. The EQUATE Statement is therefore particularly useful under the following two conditions:

Where a constant is used frequently within a program, and therefore the program would read more clearly if the constant were given a symbolic name. In the example on the facing page, "AM" is the commonly used symbol for "attribute mark", one of the standard data delimiters.

Where a MATREAD statement is used to read in an entire item from a file and disperse it into a dimensioned array. In this case, the EQUATE statement may be used to give symbolic names to the individual array elements which makes the program more meaningful. For example:

```
DIM ITEM(20)
EQUATE BIRTHDATE TO ITEM(1), SOC.SEC.NO. TO ITEM(2)
EQUATE SALARY TO ITEM(3)
```

in this case, the variables BIRTHDATE, SOC.SEC.NO. and SALARY are rendered equivalent to the first three elements of the array ITEM. These meaningful variables are then used in the remainder of the program.

STATEMENT	EXPLANATION
EQUATE PI TO 3.1416	Variable PI is compiled as the value 3.1416 at compile time.
EQUATE STARS TO "*****"	Variable STARS is compiled as the value of five asterisks at compile time.
EQUATE AM TO CHAR(254)	Variable AM is equivalent to the ASCII charater generated by the CHAR function.
EQUATE PART TO ITEM(3)	Variable PART is equivalent to element 3 of array ITEM.

Sample usage of the EQUATE statement.

9.47 EXP FUNCTION : EXPONENTIAL CAPABILITY

The EXPONENTIAL function generates the result of raising base 'e' to the power designated by the expression. (Base 'e' is 2.7183)

FORMAT:

EXP(expression)

The EXPONENTIAL function raises the number 'e' (2.7183) to the value of the expression. If the value of the expression is such that 'e' to that power is greater than 14,073,748,834, the function returns a value of zero.

The EXPONENTIAL function is the inverse of the NATURAL LOGARITHM (LN) function.

(See: LN)

STATEMENT	EXPLANATION
YY = EXP(XX)	Assigns the result of raising base 'e' the power of the expression XX, to variable YY.
PRINT EXP(1)	Prints "0"
PRINT EXP(-110+120)	Prints "2.3026"
PRINT 24 + EXP(1000)	Prints "30.9079"
PRINT EXP(10000)	Prints "9.2105"

Sample usage of the EXP function.

9.48 EXTRACT FUNCTION : DYNAMIC ARRAY EXTRACTION

The EXTRACT function returns an attribute, a value, or a secondary value from a string in 'item' format (called a dynamic array).

FORMAT:

```
EXTRACT(da.expression,att#{,value#,sub-value#})
                        or
da.expression<att#{,value#,sub-value#}>
```

the dynamic array used by this function is specified by the da.expression. Whether an attribute, a value, or a secondary value is extracted depends upon the values of the second, third, and fourth parameters. The att# specifies an attribute, the value# specifies an value, and the sub-value# specifies a secondary value. If the third and fourth parameters both have a value of 0, or have been dropped, then an entire attribute is extracted. If the sub-value# (only) has a value of 0, or been dropped, then a value is extracted. If the last three parameters are all non-zero, then a secondary value is extracted. Trailing zero value# or sub-value# mark counts are not required. Consider the following example:

```
OPEN 'TEST' TO TEST ELSE STOP 201,'TEST'
READ ITEM FROM TEST,'NAME' ELSE STOP 202,'NAME'
PRINT ITEM<,3,2>
```

These statements cause value 2 of attribute 3 of item NAME in file TEST to be printed. Consider the following example:

```
OPEN 'ACCOUNT' TO ACCOUNT ELSE STOP 201,'ACCOUNT'
READ ITEM1 FROM ACCOUNT, 'ITEM1' ELSE STOP 202,'ITEM1'
IF ITEM1<3,2,1>=25 THEN PRINT "MATCH"
```

These statements cause the message "MATCH" to be printed if secondary value 1 of value 2 of attribute 3 of item ITEM1 in file ACCOUNT is equal to 25.

STATEMENT		EXPLANATION
Y=EXTRACT(X,2,0,0) Y=X<2>	or	Assigns attribute 2 of dynamic array X to variable Y.
A=3 B=2 Q1=ARR<A,B,A+1>		Assigns secondary value 4 of value 2 of attribute 3 of dynamic array ARR to variable Q1.
IF B<3,2,1> >5 THEN PRINT MSG GOSUB 100 END		If secondary value 1 of value 2 of attribute 3 of dynamic array B is greater than 5, then the value of MSG is printed and a subroutine branch is made to statement 100.
PRINT D<25,2,0>		Prints value 2 of attribute 25 of dynamic array D.

Sample usage of the EXTRACT Function.

The FIELD function returns a sub-string from a string by specifying a delimiter character.

FORMAT:

FIELD(expression,delimiter,occurence#)

The FIELD function takes the string value of the expression and searches for a sub-string delimited by the character specified by the delimiter. The occurence# specifies which occurrence of the sub-string is to be returned. If the occurence# has a value of 1, then the FIELD function will return the sub-string from the beginning of the string up to the first occurrence of of the delimiter. For example, the statement below assigns the string value of "XXX" to the variable A:

```
A = FIELD("XXX.YYY.ZZZ.555",".",1)
```

If the occurence# has a value of 2, then the sub-string delimited by the first and second occurrence of the specified delimiter character will be returned. A value of 3 for the occurence# will return the sub-string delimited by the second and third occurence of the specified delimiter character, and so on for higher values. For example, the statement below assigns the string value "ZZZ" to variable C:

```
C = FIELD("XXX.YYY.ZZZ.555",".",3)
```

(See: COL1() and COL2() Functions)

STATEMENT	EXPLANATION
T = "12345A6789A98765A" G = FIELD(T,"A",1)	Assigns the string value "12345" to variable G.
T = "12345A6789A98765A" G = FIELD(T,"A",3)	Assigns the string value "98765" to variable G.
X = "77ABCXX" Y = "$" Z = "ABC" IF FIELD(X,Y,2)= THEN STOP	The IF statement will cause the program to terminate (i.e., the value returned by the FIELD function is "ABC", which equals the value of Z, thus making the test condition true).

Sample usage of the FIELD statement.

9.50 FOOTING STATEMENT : PAGE OUTPUT FOOTINGS

The FOOTING statement causes the specified text string to be printed at the bottom of each page.

FORMAT:

FOOTING "text 'options' {text 'options'}"

The first FOOTING statement executed will initialize the page parameters. Subsequently, the Footing literal data may be changed by a new FOOTING Statement, and the new Footing will be output when the end of the current page is reached.

The special Footing option characters listed below may be used as part of a FOOTING string expression. These special characters will be converted and printed as part of the Footing. Option characters are enclosed in single quotes. Consider, for example:

FOOTING "Copyright 1985 PICK SYSTEMS 'T' PAGE 'P'"

This statement will print a Footing consisting of: the words "Copyright 1985 PICK SYSTEMS", followed by the current time and date, followed by the word "PAGE", followed by the current page number. Page numbers are assigned in ascending order starting with page 1.

The footing literal data may be changed at any time in the PICK/BASIC program by another FOOTING statement; this change will take effect when the end of the current page is reached. The same set of special option characters are used in heading statements.

(See: HEADING)

HEADING OPTIONS	Character is Converted to:
P	Current page number
L	Carriage return/line feed
T	Current time and date
C	Centers the line
D	Current date
N	No stop at end of page

Special Option Characters for FOOTING Statement.

STATEMENT	EXPLANATION
FOOTING "TIME & DATE: 'T'	The text "TIME & DATE:" will be printed followed by the current time and date.
HEADING "PAGE 'P'"	The text "PAGE" will be printed followed by the current page number.
FOOTING "'LTP'"	The following footing will be printed: the current time, date and page.

Sample Usage of FOOTING Statements.

The FOR and NEXT statements are used to specify the beginning and ending points of a program loop. A loop is a portion of a program written in such a way that it will execute repeatedly until some test condition is met.

A FOR and NEXT loop causes execution of a set of statements for successive values of a variable until a limiting value is encountered. Such values are specified by establishing: 1) an initial value for a variable, 2) a limiting value for the variable, and 3) an increment value to be added to the value of the variable at the end of each pass through the loop. When the limit is exceeded, program control proceeds to the following body of the program.

FORMAT:

```
FOR variable = expression TO expression {STEP expression}
.
.
NEXT variable
```

The expression preceding TO specifies the initial value of the variable, the expression following TO gives the limiting value, and the optional expression following STEP gives the increment. If STEP is omitted, the increment value is assumed to be +1. The initial value expression is evaluated only once (when the FOR statement is executed). The other two expressions are evaluated on each iteration of the loop.

The function of the NEXT statement is to return program control to the beginning of the loop after a new value of the variable has been computed. Note that the variable in the NEXT statement must be the same as the variable in the FOR statement.

As an example, consider the execution of the following statements:

```
150 FOR J=2 TO 11 STEP 3
160 PRINT J+5
170 NEXT J
```

Statement 150 sets the initial value of J to 2 and specifies that J thereafter will be incremented by 3 each time the loop is performed, until J exceeds the limiting value 11. Statement 160 prints out the current value of the expression J+5. Statement 170 assigned J its next value (i.e., J=2+3=5) and causes program control to return to statement 150. Statement 160 is again executed, and statement 170 again increments J and causes the program to loop back. This process continues with J being incremented by 3 after each pass through the loop. When J attains the limiting value of 11, statement 160 will again be executed and control will pass to 170. J will again be incremented (i.e., J=11+3=14), and since 14 is greater than the limiting value of 11, the program will "fall through" statement 170 and control will pass to the next sequential statement following statement 170.

STATEMENTS	EXPLANATION
FOR A=1 TO 2+X-Y . . NEXT A	Limiting value is current value of expression 2+X-Y; increment value is +1.
FOR K=10 TO 1 STEP -1 . . NEXT K	Increment value is -1 (i.e., variable K will decrement by a value -1 for each of 10 passes through the loop).
FOR VAR= 0 TO STEP .1 . . NEXT VAR	Increment value is .1 (i.e., variable VAR will increment by a value of .1 for each of 11 passes through the loop).

Sample usage of the FOR...NEXT statement.

Optional condition clauses (WHILE and UNTIL) may be used in the FOR statement. FOR and NEXT loops may be "nested"; a nested loop is defined as a loop which is wholly contained within another loop.

EXTENDED FORMAT:

```
FOR variable = expression TO expression {STEP
                        expression}{WHILE expression}

FOR variable = expression TO expression {STEP
                        expression}{UNTIL expression}
```

The extended form of the FOR statement functions identically to the basic form, with the following additions.

If the WHILE clause is used, the specified expression will be evaluated for each iteration of the loop. If it evaluates to false (i.e., zero), then program control will pass to the statement immediately following the accompanying NEXT statement. If it evaluates to true (i.e., non-zero), the loop will re-iterate.

If the UNTIL clause is used, the specified expression will be evaluated for each iteration of the loop. If it evaluates to true (i.e., non-zero), then program control will pass to the statement immediately following the accompanying NEXT statement. If it evaluates to false (i.e., non-zero), the loop will re-iterate.

The following FOR and NEXT loop, for example, will execute until I=10 or until the statements within the loop cause variable A to exceed the value 100:

```
FOR I=1 TO 10 STEP .5 UNTIL A>100
  .
  .
  .
NEXT I
```

FOR and NEXT loops contained within the range of other FOR and NEXT loops are called nested loops. For example:

```
FOR I=1 TO 10
  FOR J=1 TO 10
  PRINT B (I,J)
  NEXT J
NEXT I
```

The above statements illustrate a two-level nested loop. The inner loop will be executed ten times for each of ten passes through the outer loop, i.e., the statement PRINT B(I,J) will be executed 100 times, causing matrix B to be printed in the following order: B(1,1), B(1,2), B(1,3),..., B(1,10), B(2,1), B(2,2),..., B(10,10).

Loops may be nested any number of levels. However, a nested loop must be completely contained within the range of the outer loop (i.e., the ranges of the loops may not cross).

STATEMENT	EXPLANATION
ST="X" FOR B=1 TO 10 UNTIL ST="XXXXX" ST=ST CAT "X" NEXT B	Loop will execute 4 times (i.e., an "X" is added to the string value of variable ST until the string equals "XXXXX").
A=20 FOR J=1 TO 10 WHILE A<25 A=A+1 PRINT J,A NEXT J	Loop will execute 5 times (i.e., variable A reaches 25 before variable J reaches 10).
A=0 FOR J=1 TO 10 WHILE A<25 A=A+1 PRINT J,A NEXT J	Loop will execute 10 times (i.e., variable J reaches 10 before variable A reaches 25).

Sample usage of the FOR...NEXT statement.
(Extended Form)

The GOSUB, COMPUTED GOSUB, RETURN, and RETURN TO statements (The RETURN and RETURN TO statements will be discussed in the next section.) provide internal subroutine capabilities for the PICK/BASIC program. A subroutine is an integral group of statements which handle a unique function or task. An internal subroutine is a subroutine that is contained within the program that calls it (i.e., before the END statement). The GOSUB statement transfers control to the subroutine).

FORMAT:

GOSUB statement-label

Upon execution of a GOSUB statement, program control is transferred to the statement which begins with the specified numeric statement-label. Execution proceeds sequentially from that statement until a RETURN or RETURN TO statement is encountered. Either of these statements transfers control back to the main program.

The Computed GOSUB statement is a combination of the Computed GOTO statement and the GOSUB statement. Control is transferred to one of several statement-labels selected by the current value of an index expression. Control returns to the statement following the computed GOSUB when a RETURN statement is executed.

FORMAT:

ON expression GOSUB statement-label, statement-label, ...

The expression is evaluated and truncated to an integer value. The result is used as an index into the list of statement-labels. A subroutine branch is executed to the statement-label selected.

If the expression evaluates to less than 1 or to a value greater than the number of statement-labels, no action is taken, that is, the statement immediately following the ON GOSUB will be executed next.

```
        ON I GOSUB 100,150,200
        * CONTROL TRANSFERS HERE AFTER RETURN FROM SUBROUTINE
        . . .                                (DIRECTLY IF I<1 OR I>3)
100     * CONTROL TRANSFERS HERE IF I=1
        . . .
        RETURN
150     * CONTROL TRANSFERS HERE IF I=2
        . . .
        RETURN
200     * CONTROL TRANSFERS HERE IF I=3
        . . .
        RETURN
```

Sample usage of the ON...GOSUB statement.

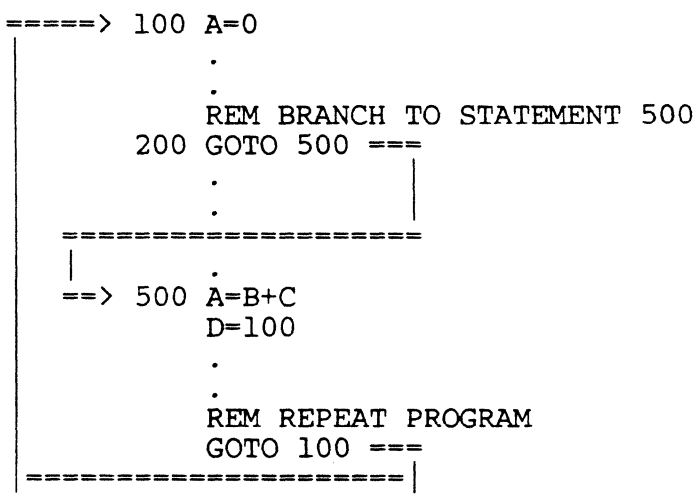
9.54 GOTO STATEMENT : UNCONDITIONAL BRANCHING

The GO{TO} statement unconditionally transfers program control to any statement within the PICK/BASIC program.

FORMAT:

GO{TO} statement-label

Execution of the GO{TO} statement causes program control to transfer to the statement which begins with the specified numeric statement-label. If a statement does not exist with the specified statement-label an error message will be printed at compile time (refer to the appendix describing compiler error messages). Note that control may be transferred to statements following the GO{TO} statement, as well as to statements preceding the GO{TO} statement.



Sample usage of the GOTO statement.

9.55 HEADING STATEMENT : PAGE OUTPUT HEADINGS

The HEADING statement causes the specified text string to be printed as the next page heading.

FORMAT:

HEADING "text 'options' {text 'options'}"

The first HEADING statement executed will initialize the page parameters. Subsequently, the Heading literal data may be changed by a new HEADING Statement, and the new Heading will be output at the beginning of the next page. The special heading option characters listed below may be used as part of a HEADING string expression. These special characters will be converted and printed as part of the heading. Option characters are enclosed in single quotes. Consider, for example:

HEADING "INVENTORY LIST 'T' PAGE 'PL'"

This statement will print a heading consisting of: the words "INVENTORY LIST", followed by the current time and date, followed by the word "PAGE", followed by the current page number, followed by a carriage return and line feed. Page numbers are assigned in ascending order starting with page 1.

The same set of special option characters are used in FOOTING statements.

(See: FOOTING)

HEADING OPTIONS	Character is Converted to:
P	Current page number
L	Carriage return/line feed
T	Current time and date
C	Centers the line
D	Current date
N	No stop at end of page

Special Option Characters for HEADING Statement.

STATEMENT	EXPLANATION
HEADING "TIME & DATE: 'TL'	The text "TIME & DATE:" will be printed followed by the current time and date plus a carriage return/line feed.
HEADING "PAGE 'PL'"	The text "PAGE" will be printed followed by the current page number and a carriage return/line feed.

Sample Usage of HEADING Statements.

The ICONV function provides the PICK input conversion capabilities to the PICK/BASIC programmer.

FORMAT:

ICONV(expression,conversion)

the conversion specifies the type of input conversion to be applied to the string value resulting from the expression. The resultant value is always a string.

(See: OCONV)

The input conversion operation specified by the conversion parameter may include any one of the following:

D Convert date to internal format (for ICONV function) or to external format (for OCONV function).

MT Converts time.

MX Convert ASCII to hexadecimal (for ICONV function) or convert hexadecimal to ASCII (for OCONV function).

T Convert by table translation.

U Call to user-defined assembly routine.

For a detailed treatment of these (and other) conversion capabilities, the user should refer to the ACCESS Chapter.

NOTE: The ACCESS 'F' conversion cannot be called by these functions. The ACCESS 'MR' or 'ML' conversion may be called by using the Format String which performs the same function and is preferable to using the ICONV or OCONV functions in this case.

STATEMENT	EXPLANATION
IDATE = ICONV("7-01-74","D")	Assigns the string value "2374" (i.e., the internal date) to the variable IDATE.
ITIME = ICONV("17:04:18","MT")	Assigns the string value "61458" (i.e., the internal time) to the variable ITIME.

Sample usage of the ICONV Function.

9.57 IF STATEMENT : SINGLE-LINE CONDITIONAL BRANCHING

The Single-Line IF statement provides the conditional execution of a sequence of PICK/BASIC statements, or the conditional execution of one of two sequences of statements.

FORMAT:

IF expression THEN statements {ELSE statements}

If the result of the test condition specified by the expression is true (i.e., non-zero), then the statement or sequence of statements following the THEN are executed. If the result of the expression is false (i.e., zero), then the statement or sequence of statements following the ELSE are executed, unless the ELSE clause is omitted, in which case control will pass to the next sequential statement following the entire IF statement. The expression may be any legal BASIC expression.

The sequence of statements in the THEN or ELSE clauses may consist of one or more statements on the same line. If more than one statement is contained in either the THEN or ELSE clause, they must be separated by semicolons. Consider the example:

```
IF ITEM THEN PRINT X;  X=X+1 ELSE PRINT X*5;  GOTO 10
```

If the current value of ITEM is non-zero (i.e., true), then this statement will print the current value of X, add one to the current value of X, and then transfer control to the next sequential instruction in the program. If the value of ITEM is zero (i.e., false), then the value of X*5 will be printed and control will transfer to statement 10.

Any statements may appear in the THEN and ELSE clauses, including additional IF statements.

The THEN clause of an IF statement is optional if the ELSE clause is present. One or the other MUST be present. This allows IF statements with the format:

IF expression ELSE statements

STATEMENT	EXPLANATION
IF A="STRING" THEN PRINT "MATCH"	Prints "MATCH" if value of A is the string "STRING".
IF X>5 THEN IF X<9 THEN GOTO 10	Transfers control to statement 10 if X is greater than 5 but less than 9.
IF Q THEN PRINT A ELSE PRINT B; STOP	The value of A is printed if Q is non-zero. If Q=0, then the value of B is printed and the program is terminated.
IF A=B THEN STOP ELSE IF C THEN GOTO 20	Program is terminated if A=B; control is passed to statement 20 if A does not equal B and if C is non-zero.

Sample usage of the Single-Line IF statement.

9.58 IF STATEMENT : MULTI-LINE CONDITIONAL BRANCHING

The Multi-Line IF statement is functionally identical to the Single-Line IF statement. It provides the conditional execution of a sequence of PICK/BASIC statements, or the conditional execution of one of two sequences of statements. The statement sequences, however, may be placed on multiple program lines.

The Multi-Line IF statement is actually an extension of the Single-Line format. With this format, the statement sequences in the THEN and ELSE clauses may be placed on multiple program lines, with each sequence being terminated by an END. The general format of the Multi-Line IF statement takes on three forms as shown in Figure A.

In each of the three forms, the ELSE clause is optional and may be included or omitted as desired. Any statements may appear in the THEN and ELSE clauses.

```
FORM 1:     IF expression THEN
            statements
            .
            .
            .
            END ELSE statements

FORM 2:     IF expression THEN
            statements
            .
            .
            .
            END ELSE
            statements
            .
            .
            .
            END

FORM 3:     IF expression THEN
            statements ELSE
            .
            .
            .
            END

NOTE:   In each of the above forms,
        the ELSE clause is optional.
```

General form of the Multi-Line IF statement.

IF STATEMENTS	EXPLANATION

IF ABC=ITEM+5 THEN
  PRINT ABC
  STOP
END ELSE PRINT ITEM; GOTO 10
	The value of ABC is printed and the program terminates if ABC=ITEM+5; otherwise the value of ITEM is printed and control passes to statement 10.

IF VAL THEN
  PRINT MESSAGE
  PRINT VAL
  VAL=100
  END
	If the value of VAL is non-zero then the value of MESSAGE is printed, the value of VAL is printed, and VAL is assigned a value of 100; otherwise control passes to the next statement following END.

10 IF S="XX" THEN PRINT "OK" ELSE
  PRINT "NO MATCH"
  PRINT S
  STOP
  END
20 REM REST OF PROGRAM
	If the value of S is the string "XX" then the message "OK" is printed and control passes to statement 20; otherwise "NO MATCH" is printed, the value of S is printed, and the program terminates.

IF X>1 THEN
  PRINT X
  X=X+1
END ELSE
  PRINT "NOT GREATER"
  GOTO 75
END
	If X>1 the value of X is printed and then incremented, and control passes to the next statement following the second END: otherwise "NOT GREATER" is printed and control passes to statement 75.

Sample usage of the Multi-Line IF statement.

9.59 INDEX FUNCTION : SEARCHING FOR SUB-STRINGS

The INDEX function searches a string for the occurrence of a sub-string and returns the starting column position of that sub-string.

FORMAT:

INDEX(string.expression,substring,occurence#)

The INDEX function takes the string value of the expression and searches for the sub-string specified by substring. The occurence# specifies which occurrence of that sub-string is sought. The resultant numeric value of the INDEX function is the starting column position of the sub-string within the string. a value of 0 is returned if the sub-string is not found.

The user should note that no blank space may appear between "INDEX" and "(". This is true for all PICK/BASIC Intrinsic Functions.

STATEMENT	EXPLANATION
A = INDEX("ABCAB","A",2)	Assigns value of 4 to variable A (i.e., 2nd occurrence of "A" is at column position 4 of "ABCAB").
X = "1234ABC" Y = "ABC" IF INDEX(X,Y,1)=5 THEN GOTO 3	The IF statement will transfer control to statement 3 (i.e., "ABC" starts at column position 5 of "1234ABC" which makes the test condition in the IF statement true).
Q = INDEX("PROGRAM","S",5)	Assigns value of 0 to variable Q (i.e., "S" does not occur in "PROGRAM").
S = "X1XX1XX1XX" FOR I=1 TO INDEX(S,"1",3) . . NEXT I	The loop will execute 8 times (i.e., 3rd occurrence of "1" appears at column position 8 of the string named S).

Sample usage of the INDEX Function.

The INPUT statement is used to request input data from the user's terminal.

FORMAT:

INPUT variable {:}

Upon execution of an INPUT statement, a "prompt" character will be printed at the user's terminal. The user's response will then be assigned to the variable indicated in the INPUT statement. For example:

INPUT A

This statement will cause a prompt character to be printed at the user's terminal. The data which the user thereupon inputs will become the current value of variable A.

A colon may be used following the input variable to supress the automatic carriage return and line feed printed when a value is input. For example:

INPUT AMOUNT:

Via the PROMPT statement, the user may select any character to be used as the input prompt character.

(See: PROMPT)

Additional functionality can be added to the input function, including masking, screen positioning, and error checking.

(See: INPUT, INPUTERR, INPUTTRAP, and INPUTNULL)

STATEMENT	EXPLANATION
INPUT VAR	Will request a value for variable VAR at the user's terminal.

Sample Usage of the INPUT Statement.

9.61 INPUT @ STATEMENT : POSITIONING MASKED INPUT

Masking functions are available for use with the INPUT statement.

FORMAT:

INPUT @(x,y):variable mask

The parameter x refers to the terminal column position, and y to the terminal row position.

This is a VERY powerful input function! It is capable of replacing as many as twenty lines of PICK/BASIC code used in screen input. Its functions include cursor addressing, output masking, editing, error messages, input masking, and exception trapping.

This command itself is used for the actual entry of the data. Ancillary functions can be performed by the commands described below. In the above example, "variable" represents the name of the variable being input, and "mask" represents a standard PICK format mask. If the variable being used already has a value it will be displayed at the specified cursor address using "mask" as the output mask. Regardless, the cursor is positioned one character back of "x" in the "@(x,y)" specification, the prompt character is printed and input is requested. If the user presses the return key, then whatever default value was there before will be accepted. Otherwise, the input will be verified against the mask, and, if acceptable, will be assigned to "variable". If the mask contains a decimal digit specification and/or a scaling factor, then numeric checking will be performed. If the mask contains a length specification (eg. R#10), then length checking will be performed. If the mask is 'D' (or any other valid date mask) then a date verification will be performed.

Note that data is converted on output and input. Thus, if you wish to input a date, the default should be stored in internal format, will be displayed and input in output format and will be placed back in the variable in internal format. Note also that the '%' is a numeric character verification symbol. Thus, for example, if the statement executed is INPUT @(20,10):SOC.SEC '%%%-%%-%%%%' and the data entered is 423-15-6897 then SOC.SEC will contain the value 423156897. If an error condition is encountered, a message is printed at the bottom of the screen.

(See: INPUT)

INPUT @(25,2):INV.DATE 'D'	Inputs a date.
INPUT @(35,7):AMOUNT 'R2,'	Inputs a dollar value.
INPUT @(20,14):NAME 'L#40'	Inputs a text field with a length specification.
INPUT @(0,10):DESC	Inputs data with no mask.

Sample usage of the INPUT @ Statement.

Some extended features of the INPUT fuction.

FORMAT:

```
INPUTERR expr

INPUTTRAP 'xx' GOTO n,n,n,n ...

INPUTTRAP 'xx' GOSUB n,n,n,n ...

INPUTNULL x
```

These are all support functions for the extended form of input statement. They allow the user to tailor the INPUT function to conform to local standards.

INPUTERR causes a message, specified by "expr", to be printed on the last line of the screen. This differs from an explicit PRINT statement in that it sets a flag indicating that a message has been printed. Thus, when the next valid entry is made the system will check the flag and clear the bottom line.

INPUTTRAP allows the user to set a trap for a particular character or characters. Each character in the string specification corresponds to a label in the GOTO or GOSUB clause. Thus, for example, if the statement INPUTTRAP '_X' GOTO 10,20 is executed, the subsequent entry of a '_' character will cause a branch to "10" and the entry of 'X' will cause a branch to "20". The GOSUB form of this expression will cause a subroutine call to be issued instead. Caution - the subroutine RETURN statement will cause a return to the statement following the INPUTTRAP statement - not the one following the INPUT statement.

The INPUTNULL statement allows the user to define a character which is to signify that whatever default value was present is to be replaced by the null string. Thus, if the statement INPUTNULL '/' is executed, the subsequent entry of a '/' character will cause a defaulted value to go to null. Note that the default character is '_'.

(See: INPUT)

INPUTERR 'INVALID DATA!'	Displays error message
INPUTTRAP '*/' GOTO 150,170	Causes branching if either '*' or '/' is entered.
INPUTNULL '@'	Causes the '@' character to null defaults in INPUT statements.

Examples of INPUTERR, INPUTTRAP and INPUTNULL Statements.

9.63 INSERT FUNCTION : DYNAMIC ARRAY INSERTION

The INSERT function inserts an attribute, a value, or a secondary value into a string in 'item' format (called a dynamic array).

FORMAT:

INSERT(da.expression,att#{,value#,sub-value#,}{;}new.expression)

The dynamic array used by this function is specified by the da.expression. Whether an attribute, a value, or a secondary value is replaced depends upon the values of the second, third, and fourth parameters. The att# specifies an attribute, the value# specifies a value, and the sub-value# specifies a secondary value. If the value# and sub-value# both have a value of 0, (or dropped) then an entire attribute is replaced. If the sub-value# (only) has a value of 0, (or dropped) then a value is replaced. If the second, third, and fourth parameters are all non-zero, then a secondary value is replaced. The replacement value is specified by the new.expression. The semi-colon (;) is used whenever value# and/or sub-value# have been dropped and the new.expression is no longer the fifth parameter.

If the att#, value# or sub-value# of the INSERT function has a value of -1, then insertion after the last attribute, last value, or last secondary value (respectively) of the dynamic array is specified. For example:

```
OPEN 'FN1' TO FN1 ELSE STOP 201,'FN1'
READ B FROM FN1,'ITEMX' ELSE STOP 202,'ITEMX'
A = INSERT(B,-1;'EXAMPLE')
WRITE A ON FN1,'ITEMX'
```

These statements insert the string value "EXAMPLE" after the last attribute of item ITEMX in file FN1.

STATEMENTS	EXPLANATION
Y = INSERT(X3,2,0,"XYZ")	Inserts before value 2 of attribute 3 of dynamic array X the string value "XYZ" (thus creating a new value), and assigns the resultant dynamic array to variable Y.
NEW = "VALUE" TEMP = INSERT(TEMP,9,0,0,NEW)	Inserts before attribute 9 of dynamic array TEMP the string value "VALUE" (thus creating a new attribute).
A = "123456789" B = INSERT(B,3,-1,0,A)	Inserts the value "123456789" after the last value of attribute 3 of dynamic array B.
Z = INSERT(W,5,1,1,"B")	Inserts the string value "B" before secondary value 1 of value 1 of attribute 5 in dynamic array W (thus creating a new secondary value), and assigns the resultant dynamic array to variable Z.

Sample usage of the INSERT Function.

9.64 INT FUNCTION : INTEGER NUMERIC VALUE

The INT function returns an integer value. An integer is a whole number.

FORMAT:

INT(expression)

The INT function returns the integer portion of the specified expression (i.e., the fractional portion of the expression is truncated). For example:

```
PRINT INT(5.37)
```

This statement causes the value 5 to be printed.

STATEMENT	EXPLANATION
A = INT(Q)	Assigns the integer value of variable Q to variable A.
A = 3.55 B = 3.6 C = INT(A+B)	Assigns the value 7 to variable C.
J = INT (5/3)	Assigns the value 1 to variable J.

Sample Usage of the INT Function.

9.65 LEN FUNCTION : GENERATING A LENGTH VALUE

The LEN function determines the length of a string.

FORMAT:

```
LEN(expression)
```

the LEN function returns the numeric value of the length of the string specified by the expression. For example:

```
A = "1234ABC"
B = LEN(A)
```

These statements assign the value of 7 to variable B.

STATEMENT	EXPLANATION
Q = LEN("123")	Assigns the value 3 to variable Q (i.e., the length of string "123").
X = "123" Y = "ABC" Z = LEN(X CAT Y)	Assigns the value 6 to variable Z.

Sample Usage of the LEN Function.

9.66 LN FUNCTION : NATURAL LOGARITHM

The NATURAL LOGARITHM function generates the natural logarithm of the expression. (Base 'e' is 2.7183)

FORMAT:

LN(expression)

The NATURAL LOGARITHM (LN) function generates the natural (base e) logarithm of the expression. If the value of the expression is less than or equal to zero, the LN function returns a value of zero. The upper range limit for the expression is 14,073,748,834.

The NATURAL LOGARITHM function is the inverse of the EXPONENTIAL function.

(See: EXP)

STATEMENT	EXPLANATION
YY = LN(XX)	Assigns the natural logarithm of expression XX to variable YY.
PRINT LN(-35+37)	Prints "0.6932"
PRINT LN(1000)	Prints "6.9079"
PRINT LN(10000)	Prints "9.2105"

Sample usage of the LN function.

The LOCATE statement may be used to find the index of an attribute, a value, or a secondary value within a dynamic array. The elements of the dynamic array may be specified as being in ascending or descending ASCII sequence, and sorted with either right or left justification. If the specified attribute, value, or secondary value is not present in the dynamic array in the proper sequence, an index value is returned which may be used in an INSERT statement to place the sought element into its proper location.

FORMAT:
LOCATE('string',item{,att#{,val#}};index#{;'sequence}) THEN/ELSE stmts

'String' is the element to be located in dynamic array 'item'. 'Index#' is the variable into which the index of 'string' is to be stored. 'Att#' and "val#" are optional parameters which restrict the scope of the search within 'item'. If neither parameter is present, 'string' is tested for equality with attributes in 'item', and 'index#' returns an attribute number. If 'att#' is present, 'string' is compared with values within the attribute specified by "att#" of "item", and "index#" returns a value number. If 'val#' is also present, the search is conducted for secondary values of the specified attribute and value of 'item', and 'index#' returns a secondary value number.

If 'sequence' has the value 'A' (or any string value beginning with 'A'), the elements of "item" are assumed to be sorted in ascending sequence. If "sequence" has the value "D" (or any string value beginning with "D"), the elements are assumed to be in descending sequence. All other values for 'sequence' are ignored.

If the first character of 'sequence' is 'A' or 'D', the second character determines the justification used when sorting the elements. If the second character is "R", right justification is used. For any other value, including null, left justification is used. If 'sequence' is not specified and the string is not found, the default will be to the last position.

SEQUENCE PARAMETERS	SEQUENCE PARAMETERS
AL - ascending, left-justified	DL - descending, left-justified
AR - ascending, right-justified	DR - descending, right-justified

STATEMENT:
```
LOCATE('55',ITEM,3,1;INDEX1;'AR') ELSE ITEM = INSERT(ITEM,3,1,INDEX1,'55')
```

EXPLANATION
The third attribute, first value of dynamic array 'ITEM' is searched for the numeric literal '55'. 'INDEX1' will return with the secondary value index if the numeric is found, and will return with the correct secondary value index if the numeric is not found. If it is not found, control passes to the ELSE clause which inserts the numeric into the correct position by virtue of the index contained in 'INDEX1'. The optional parameter 'AR' specifies ascending sequence and right justification.

Sample usage of the the LOCATE statement.

The LOCK statement provides a file and execution lock capability for PICK/BASIC programs. The LOCK statement sets execution locks while the UNLOCK statement releases them.

FORMAT:

LOCK expression {THEN/ELSE statements}

The LOCK statement sets an execution lock so that when any other BASIC program attempts to set the same lock, then that program will either execute an alternate set of statements or will pause until the lock is released (via an UNLOCK statement) by the program which originally locked it.

Execution locks may be used as file locks to prevent multiple PICK/BASIC programs from updating the same files simultaneously. There are 48 execution locks numbered from 0 through 47.

the value of the expression specifies which execution lock is to be set. If the specified execution lock has already been set by another concurrently running program (and the ELSE clause is not used), then program execution will temporarily halt until the lock is released by the other program.

If the ELSE clause is used, then the statement(s) following the ELSE will be executed if the specified lock has already been set by another program. The statements in the THEN/ELSE clause may be placed on the same line separated by semicolons, or may be placed on multiple lines terminated by an END (i.e., the THEN/ELSE clause takes on the same format as the THEN/ELSE clause in the IF statement).

All execution locks set by a program will automatically be released upon termination of the program.

(See: UNLOCK)

STATEMENTS	EXPLANATION
LOCK 15 ELSE STOP	Sets execution lock 15 (if lock 15 is already set, program terminates.
LOCK 2	Sets execution lock 2.
LOCK 10 ELSE PRINT X; GOTO 5	Sets execution lock 10 (if lock 10 is already set, the value of X is printed and program branches to statement 5.)

Sample Usage of the LOCK Statement.

9.69 LOOP STATEMENT : STRUCTURED LOOPING

| Program loops may be constructed via the use of the LOOP statement. |

FORMAT:

LOOP {statements} WHILE expression DO {statements} REPEAT

LOOP {statements} UNTIL expression DO {statements} REPEAT

Execution of a LOOP statement proceeds as follows. First the statements (if any) following "LOOP" will be executed. Then the expression is evaluated. One of the following is then performed depending upon the form used:

- If the "WHILE" form is used, then the statements following "DO" (if any) will be executed and program control will loop back to the beginning of the loop if the expression evaluates to true (i.e., non-zero), or program control will proceed with the next sequential statement following "REPEAT" (i.e., control passes out of the loop) if the expression evaluates to false (i.e., zero).

- If the "UNTIL" form is used, then the statements following "DO" (if any) will be executed and program control will loop back to the beginning of the loop if the expression evaluates to false (i.e., zero), or program control will proceed with the next sequential statement following "REPEAT" (i.e., control passes out of the loop) if the expression evaluates to true (i.e., non-zero).

Statements used within the LOOP statement may be placed on one line separated by semicolons, or may be placed on multiple lines. Consider the following example:

```
LOOP UNTIL A=4 DO A=A+1; PRINT A REPEAT
```

Assuming that the value of variable A is 0 when the LOOP statement is first executed, this statement will print the sequential values of A from 1 through 4 (i.e., the loop will execute 4 times). As a further example, consider the statement:

```
LOOP X=X-10 WHILE X>40 DO PRINT X REPEAT
```

Assuming, for example, that the value of variable X is 100 when the above LOOP statement is first executed, this statement will print the values of X from 90 down through 50 in increments of -10 (i.e., the loop will execute 5 times).

STATEMENTS	EXPLANATION
J=0 LOOP PRINT J J=J+1 WHILE J<4 DO REPEAT	Loop will execute 4 times (i.e., sequential values of variable J from 0 through 3 will be printed).
Q=6 LOOP Q=Q-1 WHILE Q DO PRINT Q REPEAT	Loop will execute 5 times (i.e., values of variable Q will be printed in the following order: 5, 4, 3, 2, and 1).
Q=6 LOOP PRINT Q WHILE Q DO Q=Q-1 REPEAT	Loop will execute 7 times (i.e., values of variable Q will be printed in the following order: 6, 5, 4, 3, 2, 1, and 0).
B=1 LOOP UNTIL B=6 DO B=B+1 PRINT B REPEAT	Loop will execute 5 times (i.e., sequential values of variable B from 2 through 6 will be printed).

Sample usage of the LOOP statement.

MAT Assignment and Copy statements are used to assign values to each element in the array.

FORMAT: MAT variable = expression

The MAT Assignment statement is similar to the Simple Assignment statement. It assigns a single value to all elements in an array.

The resultant value of the expression (which may be any legal expression) is assigned to each element of the array. The array being assigned is specified by the "variable" parameter. The specified array must have been previously dimensioned via a DIM statement. The following statement, for example, assigns the current value of X+Y-3 to each element of array A:

```
MAT A = X+Y-3
```

FORMAT: MAT variable = MAT variable

The MAT Copy statement copies one array to another. The first element of the array on the right becomes the first element of the array on the left, the second element on the right becomes the second element on the left, and so forth. Each variable name must have been dimensioned, and the number of elements in the two arrays must match; if not, an error message occurs.

Arrays are copied in row major order, i.e., with the second subscript (column) varying first. Consider the following example:

```
    Program Code            Resulting Array Values
    DIM X(5,2), Y(10)       X(1,1) = Y(1) = 1
    FOR I=1 TO 10           X(1,2) = Y(2) = 2
    Y(I)=I                  X(2,1) = Y(3) = 3
    NEXT I                         .
    MAT X = MAT Y                  .
                            X(5,2) = Y(10) = 10
```

The program dimensions two arrays as both having ten elements (5x2=10), initializes array Y elements to the numbers 1 through 10, and copies array Y to array X, giving the array elements the indicated values.

STATEMENTS	EXPLANATION
MAT TABLE=1	Assigns a value of 1 to each element of array TABLE.
MAT XYZ=A+B/C	Assigns the expression value to each element of array XYZ.
DIM A(20), B(20) . MAT A = MAT B	Dimensions two vectors of equal length, and assigns to elements of A the values of corresponding elements of B.
DIM TAB1 (10,10), TAB2(50,2) . . MAT TAB1 = MAT TAB2	Dimensions two arrays of the same number of elements (10x10=50x2), and copies TAB2 values to TAB1 in row major order.

Sample usage of the MAT Assignment and Copy statements.

9.71 MATREAD STATEMENT : MULTIPLE ATTRIBUTES

The MATREAD statement reads a file item and assigns the value of each attribute to consecutive vector elements.

FORMAT:

MATREAD array.var FROM {file.variable,} itemname THEN/ELSE statements

The MATREAD statement reads the file item specified by the itemname and assigns the string value of each attribute to consecutive elements of the vector specified by the array.variable. If the file.variable is used, the item will be read from the file previously assigned to that file.variable via an OPEN statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

If a non-existent item is specified, then the statements following the ELSE will be executed. The statements in the THEN/ELSE clause may appear on one line separated by semicolons, or on multiple lines terminated by an END (i.e., the THEN/ELSE clause takes on the same format as the THEN/ELSE clause in the IF statement). If the item does not exist, the contents of the vector remain unchanged.

If the number of item attributes is less than the DIMensioned vector size, the trailing vector elements are assigned a null string. If the number of attributes in the item exceeds the DIMensioned vector size, the remaining attributes will be assigned to the last element of the array.

(See: MATREADU)

STATEMENT	EXPLANATION
DIM ITEM (20) OPEN '', 'LOG' TO F1 ELSE STOP MATREAD ITEM FROM F1, 'TEST' ELSE STOP	Reads the item named TEST from the data file named LOG and assigns the string value of each attribute to consecutive elements of vector ITEM, starting with the first element

Sample Usage of the MATREAD Statement.

9.72 MATREADU STATEMENT : GROUP LOCKS

MATREADU provides the facility to lock a group of items in a file prior to updating an item in the group. Using a group lock prevents updating of an item by two or more programs simultameouslly while still allowing multiple program access to the file.

FORMAT:

MATREADU variable FROM {file.var,} itemname THEN/ELSE statements

This statement functions identically to the MATREAD statement, but additionally locks the group of the file in which the item to be accessed falls.

(See: MATREAD)

A group lock will prevent:

1. Access of items in the locked group of other PICK/BASIC programs using the READU, READVU, and MATREADU statements.

2. Update by any other program of any item in the locked group.

3. Access of the group by the file-save process.

The group will become unlocked when any item in that group is updated by the process which has it locked, when the PICK/BASIC program is terminated, or a RELEASE statement unlocks the group. Items can be updated to the group without unlocking it by using the WRITEU, WRITEVU or MATWRITEU statements.

Other processes (as in 1,2,3 above) which encounter a group lock will be suspended until the group becomes unlocked.

The maximum number of groups which may be locked by all processes in the system is 32. If a process attempts to lock a group when 32 locks are already set, it will be suspended until some group is unlocked.

(See: MATWRITEU)

STATEMENTS	EXPLANATION
MATREADU T FROM XM, "N4" ELSE NULL	This example shows use of a null ELSE clause to lock the group regardless of whether the item is existent or not.

Sample Usage of the MATREADU statement.

9.73 MATWRITE STATEMENT : MULTIPLE ATTRIBUTES

The MATWRITE statement writes a file item with the contents of a vector.

FORMAT:

MATWRITE array.variable ON {file.variable,} itemname

The MATWRITE statement replaces the attributes of the item specified by the itemname with the string value of the consective elements of the vector named by the array.variable. If the file.variable is used, the item will be written in the file previously assigned to that file.variable via an open statement. If the file.variable is omitted, then the internal default variable is used. If the itemname specifies an item which does not exist, then a new item will be created. The number of attributes in the item is determined by the DIMensioned size of the vector.

(See: MATWRITEU)

STATEMENT	EXPLANATION
DIM ITEM (10) OPEN '', 'TEST' ELSE STOP FOR I=1 TO 10 ITEM(I)=I NEXT I MATWRITE ITEM ON "JUNK"	Writes an item named JUNK in the file named TEST. The item written will contain 10 attributes whose string values are 1 through 10.

Sample Usage of the MATWRITE Statement.

9.74 MATWRITEU STATEMENT : UPDATE LOCKS

The MATWRITEU statement has the letter "U" appended to it to imply update. This command will not unlock the group locked by the program.

FORMAT:

MATWRITEU variable ON {file.variable,} itemname

This command executes similar to the MATWRITE statement with the following added functionality.

(See: MATWRITE)

This command will not unlock the group locked by the program. This varient is used primarily for master file updates when several transactions are being processed and an update of the master item is made following each transaction update.

If the group is not locked when the MATWRITEU statement is executed, the group will not be locked by the execution of the command.

STATEMENT	EXPLANATION
MATWRITEU ARRAY ON FILE.NAME,ID	Replaces the attributes of the item specified by ID (in the file opened and assigned to variable FILE.NAME) with the consecutive elements of vector ARRAY. Does not unlock the group.

Sample usage of the MATWRITEU statement.

9.75 NOT FUNCTION : LOGIC CAPABILITY

The NOT function returns a value of true (1) if the given expression evaluates to 0 and a value of false (0) if the expression evaluates to a non-zero quantity.

FORMAT:

```
NOT(expression)
```

The NOT function returns the logical inverse of the specified expression; it returns a value of true (i.e., generates a value of 1) if the expression evaluates to 0, and returns a value of false (i.e., generates a value of 0) if the expression evaluates to a non-zero quantity. The specified expression must evaluate to a numeric quantity or a numeric string. The following statement, for example, assigns the value 1 to the variable X:

```
X = NOT(0)
```

As a futher example, the following statements cause the value 0 to be printed:

```
A = 1
B = 5
PRINT NOT(A AND B)
```

STATEMENT	EXPLANATION
X=A AND NOT(B)	Assigns the value 1 to variable X if current value of variable A is 1 and current value of variable B is 0. Assigns a value of 0 to X otherwise.
IF NOT(X1)THEN STOP	Program terminates if current value of variable X1 is 0.
PRINT NOT(M) OR NOT(NUM(N))	Prints a value of 1 if current value of variable M is 0 or current value of variable N is a non-numeric string. Otherwise prints a zero.

Sample usage of the NOT Function.

9.76 NULL STATEMENT : NON-OPERATION

The NULL statement specifies a non-operation, and may be used anywhere in the program where a PICK/BASIC statement is required.

FORMAT:

```
...NULL...
```

The NULL statement is used in situations where a PICK/BASIC statement is required, but no operation or action is desired. Consider the following example:

```
IF X1 MATCHES "9N" THEN NULL ELSE GOTO 100
```

This statement will cause program control to branch to statement 100 if the current string value of variable X1 does not consist of 9 numeric characters. If the current string value of variable X1 does consist of 9 numeric characters, then no action will be taken and program control will proceed to the next sequential PICK/BASIC statement.

The NULL statement may be used anywhere in the PICK/BASIC program where a statement is required.

STATEMENT	EXPLANATION
10 NULL	This statement does not result in any operation or action; however, since it is preceded by a statement label (10) it may be used as a program entry point for GOTO or GOSUB stmts elsewhere in the program.
IF A=0 THEN NULL ELSE PRINT "A NON-ZERO" GOSUB 45 STOP END	If the current value of variable A is non-zero, then the sequence of statements following the ELSE will be executed. If A=0, no action is taken and control passes to the next sequential statement following the END.
READ A FROM "ABC" ELSE NULL	File item ABC is read and assigned to variable A. If ABC does not exist, no action is taken. (Refer to description of READ statement for further information).

Sample usage of the NULL statement.

9.77 NUM FUNCTION : NUMERIC STRING DETERMINATION

The NUM function returns a value of true (1) if the given expression evaluates to a number or a numeric string.

FORMAT:

NUM(expression)

The NUM function tests the given expression for a numeric value. For example, if the expression evaluates to a number or numeric string the NUM function will return a value of true (i.e., generating a value of 1).

Inversely, an expression evaluating to a letter or an alphabetic string will cause the NUM function to return a value of false (0). Consider the following example:

```
IF NUM(expression) THEN PRINT "NUMERIC DATA"
```

This statement will print the text "NUMERIC DATA" if the current value of variable "expression" is a number or a numeric string. In the case of a non-numeric, non-alphabetic character or string (#, ?, ., etc.) a value of false would be returned for both the NUM and ALPHA functions. The empty string ('') is considered to be a numeric string, but not an alphabetic string.

(See: ALPHA)

STATEMENT	EXPLANATION
A1=NUM(123)	Assigns a value of 1 to variable A1.
A2=NUM("123")	Assigns a value of 1 to variable A2.
A3=NUM("12C")	Assigns a value of 0 to variable A3.

Sample Usage of the NUM Function.

The OCONV function provides the PICK output conversion capabilities to the PICK/BASIC programmer.

FORMAT:

OCONV(expression,conversion)

the conversion specifies the type of output conversion to be applied to the string value resulting from the expression. The resultant value is always a string.

(See: ICONV)

The output conversion operation specified by the conversion parameter may include any one of the following:

D Convert date to internal format (for ICONV function) or to external format (for OCONV function).

MT Converts time.

MX Convert ASCII to hexadecimal (for ICONV function) or convert hexadecimal to ASCII (for OCONV function).

T Convert by table translation.

U Call to user-defined assembly routine.

For a detailed treatment of these (and other) conversion capabilities, the user should refer to the ACCESS chapter.

NOTE: The ACCESS 'F' conversion cannot be called by these functions. The ACCESS 'MR' or 'ML' conversion may be called by using the Format String which performs the same function and is preferable to using the ICONV or OCONV functions in this case.

STATEMENT	EXPLANATION
A = "2374" B = "D" XDATE = OCONV(A,B)	Assigns the string value "01 JUL 1974" (i.e., the external date) to the variable XDATE.
A = OCONV(0, 'U50BB') PRINT A END	Assigns the string value of the line number and using account name to A. "02 SYSPROG" is printed.

Sample Usage of the OCONV Function.

The ON GOTO statement transfers control to one of several statement-labels selected by the current value of an index expression.

FORMAT:

ON expression GOTO statement-label, statement-label,...

Upon execution of the ON GOTO statement, program control is transferred to the statement which begins with the numeric statement-label selected by the expression. Statement-labels in the list are numbered 1, 2, 3,.... In executing the ON GOTO statement, the expression is evaluated and then the result of the expression is truncated to an integer value.

Consider the following example:

```
ON I GOTO 50, 100, 150
.
.
50  . . .
.
100 . . .
.
150 . . .
.
```

The labels in the label list may precede or follow the ON GOTO statement. If the current value of variable I=1, control transfers to the first statement-label, i.e., the statement with label 50. If I=2, control transfers to the third statement-label, i.e., statement 150.

If the value of the expression evaluates to less than one or greater than the number of statement-labels, no action is taken, that is, the statement immediately following the ON GOTO will be executed next.

STATEMENT	EXPLANATION
ON M+N GOTO 40, 61, 5, 7	Transfer control to statement 40, 61, 5, or 7 depending on the value of M+N being 1, 2, 3, or 4 respectively.
ON C GOTO 25, 25, 20	Transfer control to statement 25 if C= 1 or 2, to statement 20 in all other cases.
IF A GE 1 AND A LE 3 THEN ON A GOTO 110, 120, 130 END	The IF statement assures that A is in range for the computed GOTO statement.

Sample usage of the ON...GOTO statement.

9.80 OPEN STATEMENT : OPENING I/O FILES

The OPEN statement is used to select a file for subsequent input, output, or update. Before a file can be accessed by a READ, WRITE, DELETE, MATREAD, MATWRITE, READV, or WRITEV etc. statement, it must be opened via an OPEN statement.

FORMAT:

OPEN {"DICT,"}, "expression" {TO variable} THEN/ELSE statements

The expression in the OPEN statement indicates the file name. If the first parameter is a 'D' or any string beginning with D (such as 'DICT') then the dictionary section of the file is opened. The word DICT must be explicitly supplied to open a dictionary level file. If the file is a multiple data file (that is, multiple data files associated with a single dictionary), to open one of the data sections the format: 'dictname,dataname' is used.

If the "TO variable" option is used, then the dictionary or data section of the file will be assigned to the specified variable for subsequent reference. If the "TO variable" option is omitted, then an internal default variable is generated; subsequent I/O statements not specifying a file variable will then automatically default to this file.

If the file indicated in the OPEN statement does not exist, then the statement or sequence of statements following the ELSE will be executed. The statements in the ELSE clause may be placed on the same line separated by semicolons, or may be placed on multiple lines terminated by an END (i.e., the ELSE clause takes on the same format as the ELSE clause in the IF statement).

There is no limit to the number of files that may be open at any given time.

STATEMENT	EXPLANATION
A='DICT' OPEN A, 'XYZ' TO B ELSE PRINT "NO XYZ" STOP END	Opens the dictionary portion of file XYZ and assigns it to variable B. If XYZ does not exist, the text "NO XYZ" is printed and the program terminates.
OPEN '','ABC,X' TO D5 ELSE STOP	Opens data section X of file ABC and assigns it to variable D5. If ABC,X does not exist, program terminates.
X='' Y='TEST1' Z='NO FILE' OPEN X, Y ELSE PRINT Z; GOTO 5	Opens data section of file TEST1 and assigns it to internal default variable. If TEST1 does not exist, "NO FILE" is printed and control passes to statement 5.

Sample usage of the OPEN statement.

The PAGE statement causes the current output device to page, and causes the heading specified by the most recent HEADING statement to be printed as a page heading. The page number may optionally be reset by the PAGE statement.

FORMAT:

PAGE {expression}

The PAGE statement causes the current output device to page, and causes the heading specified by the most recent HEADING statement to be printed at the top of the page. the number of print lines per page is controlled by the current TERM command (see TERM - TCL section). if a Footing statement has also been used, the PAGE statement will cause the footing to be printed out at the bottom of the page. If only a footing is desired, a null heading should be assigned. Headings and/or footings must be assigned before the PAGE statement is encountered.

If the PAGE statement has the optional expression, the expression is evaluated and the resulting number becomes the next page number used. If a FOOTING is in effect at the time that the page number is changed, the footing will be printed with a page number one less than the evaluated expression!

STATEMENT	EXPLANATION
HEADING "ANNUAL STATISTICS" FOOTING "XYZ CORPORATION" PAGE	The PAGE statement will cause both the specified heading and footing to be printed out when the paging is executed.
PAGE 1	This statement will cause the current footing, if any, to print (with a page number of 0), and the current heading, if any, to print with a page number of 1.
PAGE X+Y	The current footing and heading will be output, and the page number set to the evaluated result of X+Y.

Sample Usage of PAGE Statement.

9.82 PRECISION DECLARATION : SELECTING NUMERIC PRECISION

The PRECISION declaration allows the user to select the degree of precision to which all values are calculated within a given program.

FORMAT:

PRECISION n

n is a number from 0-4.

The default precision value is 4, that is, all values are stored in an internal form with 4 fractional places, and all computations are performed to this degree of precision. The desired number of fractional digits may be specified by a PRECISION declaration within the range of 0-4.

Only one PRECISION declaration is allowed in a program. If more than one is encountered, a warning message is printed and the declaration is ignored.

Where external subroutines are used, the mainline program and all external subroutines must have the same PRECISION. If the precision is different between the calling program and the subroutine, a warning message will be printed.

Changing the precision changes the acceptable form of a number; a number is defined as having a maximum of "n" fractional digits, where "n" is the precision value. Thus, the value:

1234.567

is a legal number if the precision is 3 or 4, but is not a legal number if the precision is 0, 1 or 2.

Setting a precision of zero implies that all values are treated as integers.

STATEMENT	EXPLANATION
PRECISION 0 A = 3 B = A/2	All numeric values in the program will be treated as integers. The value returned for B will be 1, not 1.5.
PRECISION 1	All numeric values in the program will be calculated to one fractional digit.
PRECISION 2	All numeric values in the program will be calculated to two fractional digits.
PRECISION 3	All numeric values in the program will be calculated to three fractional digits.

Sample Usage of PRECISION Declaration.

The PRINT statement outputs data to the device selected by the PRINTER statement. The PRINT ON option allows output to multiple print files.

FORMAT:

PRINT {ON expression} print-list

The PRINT statement without the ON option is used to output variable or literal values to the terminal or line printer, as previously selected by a PRINTER statement. The print-list may consist of a single expression, or a series of expressions, separated by commas or colons (these punctuation marks are used to denote output formatting; refer to the section Tabulation and Concatenation in PRINT Statement). The expressions may be any legal PICK/BASIC expressions. The following statement, for example, will print the current value of the expression X+Y:

```
PRINT X+Y
```

The PRINT ON statement (i.e., with the ON option) is used, when PRINTER ON is in effect, to output the print-list items to a numbered print file.This is usually done when building several reports at the same time, each having a different number. The expression following ON indicates the print file number, which may be from 0 to 254 (selected arbitrarily by the program). Consider the following example:

```
PRINT ON 1 A,B,C,D
PRINT ON 2 E,F,G,H
PRINT ON 3 X,Y,Z
```

These statements will generate 3 separate output listings, one containing A, B, C, and D values, one containing E, F, G, and H values, and the third containing X, Y and Z values.

When the ON expression is omitted, print file zero is used.

The HEADING statement affects only print file zero. Pagination must be handled by the program for print files other than zero. Lack of pagination will result in continuous printing across page boundaries.

When PRINTER OFF is in effect, both PRINT ON and PRINT operate identically, i.e., all output is to the terminal. The contents of all print files used by the program, including print file zero, will be output to the printer in sequence when a PRINTER CLOSE statement is given or on termination of the program.

STATEMENT	EXPLANATION
PRINTER ON PRINT X	Causes the value of X to be output to print file 0.
PRINTER ON PRINT ON 24 X	Causes the value of X to be output to print file 24.
N=50 PRINT ON N X,Y,Z	Outputs print-list to print file 50.
PRINTER ON PRINT ON 15 "100" PRINT ON 40 "100"	Causes the value 100 to be copied to both print file 15 and print file 40.
PRINTER ON PRINT A PRINT B	Print file 0 will contain the values of A and B.
PRINTER ON PRINT ON 10 F1,F2,F3 PRINT ON 20 M,N,P PRINT ON 10 F4,F5,F6	Print file 10 will contain the values of F1 through F6; print file 20 will contain the values M, N and P.

Sample usage of the PRINT statement.

The print-list of the PRINT statement may specify tabulation or concatenation when printing multiple items.

Output values may be aligned at tab positions across the output page by using commas to separate the print-list expressions. Tab positions are pre-set at every 18 character positions. Consider the following example:

```
PRINT (50*3)+2, A, "END"
```

Assuming that the current value of A is 37, this statement will print the values across the output page as follows:

```
152               37                END
```

Output values may be printed continuously across the output page by using colons to separate the print-list expressions. The following statement, for example, will cause the text message "THE VALUE OF A IS 5010" to be printed:

```
PRINT "THE VALUE OF A IS" :50:5+5
```

After the entire print-list has been printed, a carriage return and a line feed will be executed, unless the print-list ends with a colon. In that case the next value in the next PRINT statement will be printed on the same line as the very next character position. For example, these statements:

```
PRINT A:B,C,D:
PRINT E,F,G
```

will produce exactly the same output as this statement:

```
PRINT A:B,C,D:E,F,G
```

STATEMENT	EXPLANATION
PRINT A:B: PRINT C:D: PRINT E:F	Prints the current values of A, B, C, D, E, and F contiguously across the output page, each value concatenated to the next.
PRINT A=1	Prints 1 if "A=1" is true; prints 0 otherwise.
PRINT A*100,Z	Prints the value of A*100 starting at column position 1; prints the value of Z on the same line starting at column position 18 (i.e., 1st tab position).
PRINT	Prints an empty (blank) line.
PRINT "INPUT":	Prints the text "INPUT" and does not execute a carriage return or line feed.
PRINT " ", B	Prints the value of B starting at column position 18 (i.e., 1st tab position).

Sample usage of the PRINT statement formatting.

9.85 PRINTER ON/OFF STATEMENTS : SELECTING OUTPUT DEVICE

The PRINTER statement selects either the user's terminal or the line printer for subsequent program output.

FORMAT:

```
PRINTER ON
PRINTER OFF
PRINTER CLOSE
```

The PRINTER ON statement directs program output data specified by subsequent PRINT, HEADING, or PAGE statements to be output to the line printer. The PRINTER OFF statement directs subsequent program output to the terminal.

Once executed, a PRINTER ON or PRINTER OFF statement will remain in effect until a new PRINTER ON or PRINTER OFF statement is executed. If a PRINTER ON statement has not been executed, output will be to the terminal.

When a PRINTER ON statement has been issued, subsequent output data (specified by PRINT, HEADING, of PAGE statements) are not immediately printed on the line printer (Unless immediate printing is forced via the system SP-ASSIGN I or N option, as described in the PICK Peripheral Manual). Rather, the data are stored in an intermediate buffer area and are automatically printed upon termination of program execution.

If the user's application requires that the data be printed on the line printer prior to program termination, he may issue a PRINTER CLOSE statement. The PRINTER CLOSE statement will cause all data currently stored in the intermediate buffer area to immediately be printed.

When a PRINTER OFF statement has been issued, subsequent output data are always printed at the user's terminal immediately upon execution of the PRINT, HEADING, or PAGE statements (i.e., the PRINTER CLOSE statement applies only to output data directed to the line printer).

STATEMENT	EXPLANATION
PRINTER ON PRINT A PRINTER OFF PRINT B	Causes the value of variable B to be immediately printed at the user's terminal, and the value of variable A to be printed on the line printer when the program is finished executing.
PRINTER ON PRINT A PRINTER CLOSE PRINTER OFF PRINT B	Causes the value of variable A to be immediately printed on the line printer, and thereafter causes the value of variable B to be printed at the user's terminal.
PRINTER ON PRINT A PRINTER OFF PRINT B PRINTER CLOSE	Causes the value of variable B to be immediately printed at the user's terminal, and thereafter causes the value of variable A to be printed on the line printer.

Sample usage of the PRINTER ON/OFF/CLOSE statements.

The PROMPT statement is used to select the "prompt character" which is printed at the terminal to prompt the user for input.

FORMAT:

PROMPT expression

The value of the expression becomes the prompt character. For example:

PROMPT ":"

This statement selects the character ":" as the prompt character for subsequent INPUT statements. If the value of the expression is a numeric value of more than 1 digit, or a string consisting of one character, only the most significant character will be used.

When a PROMPT statement has been executed, it will remain in effect until another PROMPT statement is executed. If a PROMPT statement has not been executed, the INPUT statement will use a question mark (?) as the prompt character (i.e., "?" is the default prompt character).

(See: INPUT)

STATEMENT	EXPLANATION
PROMPT "@"	Specifies that the character @ will be used as a prompt character for subsequent INPUT statements.
PROMPT A	Specifies that the current value of A will be used as a prompt character.

Sample Usage of the PROMPT Statement.

9.87 PWR FUNCTION : RAISING BY A POWER

The PWR function raises an expression by the power parameter.

FORMAT:

PWR(expression,power)

The POWER function raises the expression to the power denoted by the power parameter. If the power parameter is zero, the function will return the value one.

If the expression raised to the power denoted by the power parameter is greater than 14,073,748,834, the function will return unpredictable numbers. If the expression is zero and the power parameter is any number other than zero, the function will return a value of zero.

Note: another way to express the PWR function is X^Y where X is raised to the Y power.

STATEMENT	EXPLANATION
YY = PWR(XX,ZZ)	Assigns the result of raising XX by the power of ZZ to the variable YY.
PRINT PWR(3+4,10)	Prints "282475249"
PRINT 6 + PWR(2,4)	Prints "22"
PRINT PWR(0,5)	Prints "0"

Sample usage of the PWR function.

9.88 READ STATEMENT : ACCESSING FILE ITEMS

The READ statement reads a file item and assigns its value to a variable.

FORMAT:

READ variable FROM {file.variable,} itemname THEN/ELSE statements

The READ statement reads the file item specified by the itemname and assigns its string value to the variable. The file.variable is optional and specifies the file variable. If the file.variable is used, the item will be read from the file previously assigned to that file.variable via an OPEN statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

If the itemname specifies the name of an item which does not exist, then the statement or sequence of statements following the ELSE will be executed. The statements in the THEN/ELSE clause may appear on one line separated by semicolons, or on multiple lines terminated by an END (i.e., the THEN/ELSE clause takes on the same format as the THEN/ELSE clause in the IF statement).

The user should note that the PICK/BASIC program will abort with an appropriate error message if the specified file has not been opened prior to the execution of the READ statement.

STATEMENT	EXPLANATION
READ A1 FROM X,"ABC" ELSE PRINT "NOT ABC" GOTO 70 END	Reads item ABC from the file opened and assigned to file variable X, and assigns its value to variable A1. If ABC does not exist, the text "NOT ABC" is printed and control passes to statement 70.
A="TEST" B="1" READ X FROM C,(A CAT B) ELSE STOP	Reads item TEST1 from the file opened and assigned to file variable C, and assigns its value to variable X. Program terminates if TEST1 does not exist.
READ Z FROM "Q" ELSE PRINT X; STOP	Reads item Q from the file opened without a file variable and assigns its value to variable Z. Prints value of X and terminates program if Q does not exist.

Sample usage of the READ statement.

The READNEXT statement reads the next Item-id from a selected list. If multiple files have been selected, which list is specified by select.variable.

FORMAT:

READNEXT variable {,vmc}{FROM select.variable} THEN/ELSE statements

The READNEXT statement reads the next Item-id and assigns its string value to the variable indicated. The Item-id is read from the list created by the most recent program SELECT statement or SELECT, SSELECT, or QSELECT command issued at the TCL level. If the list of Item-id's has been exhausted, or if no selection has been performed, the statements following the ELSE will be executed. The statements in the THEN/ELSE clause may be placed on the same line separated by semicolons, or may be placed on multiple lines terminated by an END (i.e., the THEN/ELSE clause takes on the same format as the THEN/ELSE clause in the IF statement).

READNEXT FORMATS:

READNEXT variable THEN/ELSE statements
This will read the next Item-id of the last file selected without a select.variable.

READNEXT variable,vmc THEN/ELSE statements
The 'vmc' is used for the value mark count to be obtained from the Exploding Sort (External SSELECT).

READNEXT variable FROM select.variable THEN/ELSE statements
Reads the next Item-id of the file (or variable) selected and assigned to the select.variable.

READNEXT variable,vmc FROM select.variable THEN/ELSE statements
This is a combination of the previous two forms.

READNEXT A FROM X ELSE STOP	Specifies the list selected and assigned to the select-variable X. Assigns the value of that list's next item-id to variable A. If item-id list exhausted (or if no SELECT, SSELECT or QSELECT executed), program will terminate.

READNEXT X2 ELSE
  PRINT "UNABLE"
  GOTO 50
  END
	Specifies the last list selected without a select-variable. Assigns the value of the next item-id to variable X2. If unable to read, "UNABLE" is printed and control transfers to statement 50.

FOR X=1 TO 10 READNEXT B(X) ELSE STOP NEXT X	Reads next ten item-id's and assigns values to matrix elements X(1) through X(10).

Sample usage of the READNEXT statements.

BASIC programs may specify Magnetic Tape I/O operations through the use of the READT (Read Tape Record) statement. The record length on the tape is as specified by the most recent T-ATT statement executed at the TCL level.

FORMAT:

READT variable THEN/ELSE statements

The READT statement reads the next record from the magnetic tape unit. The next record is read and its string value is assigned to the variable indicated. If the tape unit has not been attached, or if an End-of-File (EOF) mark is read, then the statement or sequence of statements following the ELSE will be executed.

(See: T-ATT and WRITET)

STATEMENT	EXPLANATION

READT B ELSE
    PRINT "NO"
    GOTO 5
END
	The next tape record is read and its value assigned to variable B. If EOF is read (or tape unit not attached), then "NO" is printed and control passes to statement 5.

Sample usage of the READT statement.

READU and READVU provide the facility to lock a group of items in a file prior to updating an item in the group. Using a group lock prevents updating of an item by two or more programs simultameouslly while still allowing multiple program access to the file.

FORMAT:

READU variable FROM {file.var,} itemname THEN/ELSE statements

READVU variable FROM {file.var,} itemname,att# THEN/ELSE statements

These statements function identically to the READ and READV statements, but additionally lock the group of the file in which the item to be accessed falls.

(See: READ and READV)

A group lock will prevent:

1. Access of items in the locked group of other PICK/BASIC programs using the READU, READVU, and MATREADU statements.

2. Update by any other program of any item in the locked group.

3. Access of the group by the file-save process.

The group will become unlocked when any item in that group is updated by the process which has it locked, when the PICK/BASIC program is terminated, or a RELEASE statement unlocks the group. Items can be updated to the group without unlocking it by using the WRITEU, WRITEVU or MATWRITEU statements.

Other processes (as in 1,2,3 above) which encounter a group lock will be suspended until the group becomes unlocked.

The maximum number of groups which may be locked by all processes in the system is 32. If a process attempts to lock a group when 32 locks are already set, it will be suspended until some group is unlocked.

STATEMENTS	EXPLANATION
READU ITEM FROM INV, S5 ELSE GOSUB 4	Lock group of items containing item S5. Read S5 to variable ITEM or, if S5 is non-existent, execute the ELSE clause; in either case the group remains locked until one of its items is updated, or a RELEASE statement unlocks the group.
READVU ATT FROM B, "REC", 6 ELSE STOP	Lock group of items containing item REC. Read attribute 6 to variable ATT or, if REC is non-existent, execute the ELSE clause. The group remains locked as above.

Sample Usage of READU and READVU statements.

9.92 READV STATEMENT : ACCESSING AN ATTRIBUTE

The READV statement is used to read a single attribute value from an item in a file.

FORMAT:

READV variable FROM {file.variable,} itemname,att# THEN/ELSE statements

The READV statement reads the attribute specified by att# (attribute number) from the item specified by the itemname, and assigns its string value to the variable.

The file.variable is optional and specifies the file variable; if it is used, the attribute will be read from the file previously assigned to that file.variable via an OPEN statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

If a non-existent item is specified, the statement or sequence of statements following the ELSE will be executed. The statements in the THEN/ELSE clause may be placed on the same line separated by semicolons, or may be placed on multiple lines terminated by END (i.e., the THEN/ELSE clause takes on the same format as the THEN/ELSE clause in the IF statement).

The PICK/BASIC program will abort with an appropiate error message if the specified file has not been opened prior to the execution of the READV statement.

STATEMENT	EXPLANATION
READV X FROM A, "TEST",5 ELSE PRINT ERR GOTO 70 END	Reads 5th attribute of item TEST (in the file opened and assigned to variable A) and assigns value to variable X. If item TEST is non-existent, then value of ERR is printed and control passes to statement 70.

Sample usage of the READV statements.

9.93 RELEASE STATEMENT : RELEASING GROUP UPDATE LOCKS

The RELEASE statement unlocks specified groups or all groups locked by the program.

FORMAT:

RELEASE {{file.variable,} expression}

The RELEASE statement unlocks the group hashed into by the item-id specified by the expression. If the file.variable is used, the file will be the one previously assigned to that file.variable via on OPEN statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

If the RELEASE statement is used without a file.variable or expression all groups which have been locked by the program will be unlocked.

The RELEASE statement is useful when an abnormal condition is encountered during multiple file updates. A typical sequence is to mark the item with an abnormal status, update it to the file and then RELEASE all other locked groups. This version of the RELEASE statement will release all groups locked by the program.

(See: READU, READVU and MATREADU)

STATEMENT	EXPLANATION
RELEASE	Releases all groups locked by the program.
RELEASE CUST.FILE, PART.NO	Releases group hashed into by item-id contained in PART.NO in file CUST.FILE.

Sample usage of the RELEASE statement.

The REM or MOD function generates the remainder of one number divided by another.

FORMAT:

REM(numerator,denominator)
or
MOD(numerator,denominator)

This function returns the remainder of the value of the numerator divided by the value of the denominator.

The REM and MOD (modulo) functions are identical.

STATEMENT	EXPLANATION
A = MOD(Q,Z)	Assigns the remainder of variable Q divided by Z to variable A.
A = 600 B = REM(A,-1000)	Assigns the value 600 to variable B.
J = REM(5,3)	Assigns the value 2 to variable J.
Q = MOD (1023,256)	Assigns the value 255 to the variable Q.

Sample Usage of the REM or MOD function.

9.95 REPLACE FUNCTION : DYNAMIC ARRAY REPLACEMENT

The REPLACE function replaces an attribute, a value, or a secondary value in a string in 'item' format (called a dynamic array).

FORMAT:

REPLACE(da.expression,att#{,value#,sub-value#,}{;}new.expression)

or

da.expression<att#{,value#,sub-value#}> = new.expression

The second form above is actually an extract function being utilized as a replacement function. The dynamic array used by this function is specified by the da.expression. Whether an attribute, a value, or a secondary value is replaced depends upon the values of the second, third, and fourth parameters. The att# specifies an attribute, the value# specifies a value, and the sub-value# specifies a secondary value. If the value# and sub-value# both have a value of 0, (or dropped) then an entire attribute is replaced. If the sub-value# (only) has a value of 0, (or dropped) then a value is replaced. If the second, third, and fourth parameters are all non-zero, then a secondary value is replaced. The replacement value is specified by the new.expression. The semi-colon (;) is used whenever value# and/or sub-value# have been dropped and the new.expression is no longer the fifth parameter.

If the att#, value# or sub-value# of the REPLACE function has a value of -1, then insertion after the last attribute, last value, or last secondary value (respectively) of the dynamic array is specified. For example:

```
OPEN 'XYZ' TO XYZ ELSE STOP 201,'XYZ'
READ B FROM XYZ,'ABC' ELSE STOP 202,'ABC'
B<3,-1>='NEW VALUE'
WRITE B ON XYZ,'ABC'
```

These statements insert the string value "NEW VALUE" after the last value of attribute 3 of item ABC in file XYZ.

STATEMENT	EXPLANATION
X=REPLACE(X,4;'')	Replaces attribute 4 of dynamic array X with the empty (null). string.
Y=REPLACE(X,4,0,0,'')	Replaces attribute 4 of dynamic array X with the empty (null) string. string, and assigns the resultant dynamic array to Y.
VALUE="TEST STRING" DA<4,3,2>=VALUE	Replaces secondary value 2 of value 3 of attribute 4 in dynamic array DA with the string value "TEST STRING".
X="ABC123" Y<1,1,-1>=X	Inserts the value "ABC123" after the last secondary value of value 1 of attribute 1 in dynamic array Y.

Sample usage of the REPLACE Function.

The RETURN or RETURN TO statements return control to the main program.

FORMAT:

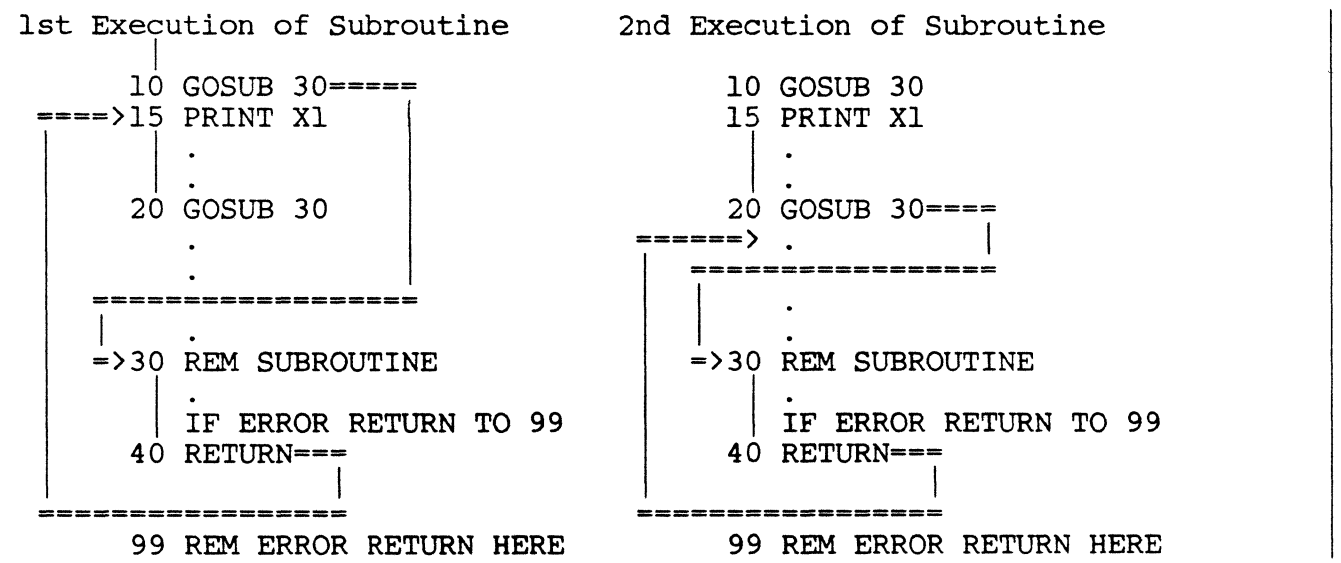
```
RETURN
RETURN TO statement-label
```

The RETURN statement will transfer control from the subroutine back to the statement immediately following the GOSUB statement. The RETURN TO statement returns control from the subroutine to the statement within the PICK/BASIC main program having the specified statement-label.

The statements in a subroutine may be any PICK/BASIC statements, including another GOSUB statement. To insure proper flow of control, each subroutine must return to the calling program by using a RETURN (or RETURN TO) statement, not a GOTO statement. The user should also insure that the subroutine cannot be executed by any flow of control other than through the execution of a GOSUB statement.

If the RETURN TO statement refers to a statement-label which is not present in the program, an error message will be printed at compile time (refer to APPENDIX C - PICK/BASIC COMPILER ERROR MESSAGES).

Consider the statements shown in the example below . Upon execution of statement 10, control will transfer to statement 30 as illustrated in the left side of the figure. The statements within the subroutine will be executed and statement 40 will then return control to statement 15. Execution will then proceed sequentially to statement 20, whereby control will again be transferred to the subroutine as shown in the right side of the figure. The conditional RETURN TO path is taken instead of the normal RETURN if the logical variable ERROR is true (=1).



Sample usage of the RETURN statement.

9.97 REWIND STATEMENT : REWINDING THE TAPE

BASIC programs may specify Magnetic Tape to rewind to the BOT mark through the use of the REWIND (Rewind Tape Unit) statement.

FORMAT:

REWIND THEN/ELSE statements

The REWIND statement rewinds the magnetic tape unit to the Beginning-of-Tape (BOT). If the tape unit has not been attached, then the statement(s) following the ELSE will be executed.

STATEMENT	EXPLANATION
REWIND ELSE STOP	Tape is rewound to BOT.

Sample Usage of the REWIND statement.

The RND function returns a random number. The range of the random number generated is controlled by the expression.

FORMAT:

RND(expression)

The RND function generates a numeric value for a random number between zero and the number specified by the expression less one (inclusive), which must be positive.

Therefore, an expression parameter which evaluates to 3, would randomly generate 0, 1, or 2. This is an invaluable function when programming games of chance.

STATEMENT	EXPLANATION
Z = RND(11)	Assigns a random number between 0 and 10 (inclusive) to the variable Z.
R = 100 Q = 50 B = RND(R+Q+1)	Assigns a random number between 0 and 150 (inclusive) to the variable B.
Y = RND(ABS(051))	Assigns a random number between 0 and 50 (inclusive) to the variable Y.

Sample Usage of the RND Function.

The SELECT command provides a facility to select a set of item-ids or attributes which, when used in conjunction with the READNEXT statement (see section on READNEXT following), may be used to access single or multiple file item-ids or attributes within a PICK/BASIC program.

FORMAT:

SELECT {file.variable}{TO select.variable}

The SELECT statement builds the same list of item-ids as a SELECT command executed at the TCL level without any selection criteria (see ACCESS). If the file.variable is used, a list of item-ids will be created for the file or item previously assigned to that file.variable via an OPEN or READ statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file.variable).

SELECT FORMATS:

SELECT
Creates a select list of item-ids from the file most recently opened without a file variable.

SELECT file.variable

Creates a select list of item-ids from the file opened to 'file-variable'.

SELECT var
Creates a select list from the attributes of the variable 'var'. Note: The select list will only include the first value of a multivalued attribute.

SELECT TO select.variable
Creates a select list from the file most recently opened without a file variable and assign the selected list to 'select-variable'.

SELECT file.variable TO select.variable
Creates a select list from the file opened to 'file-variable' and assign the selected list to 'select-variable'.

SELECT var TO select.variable
As above, except the selected list is assigned to 'select-variable'.

STATEMENT	EXPLANATION
SELECT	Builds list of item-id's using the default variable of the last file opened without a file-variable.
SELECT BP TO BLIST	Builds a list of item-ids for the file opened and assigned to file-variable 'BP'. Assigns the list to select-variable 'BLIST'.
READ A FROM FILEX,'ALIST' ELSE STOP SELECT A	Creates a select list of the attributes in item ALIST.

Sample usage of the SELECT statements.

The SEQ function converts an ASCII character to its corresponding numeric value.

FORMAT:

SEQ(expression)

The first character of the string value of the expression is converted to its corresponding numeric value. The following example will print the number 49:

```
PRINT SEQ('1')        (character 1 = 31 hex or 49 decimal)
```

Conversely, the CHAR function is available to convert a numeric expression to its corresponding ASCII character string value.

(See: CHAR)

NOTE: For a complete list of ASCII codes, refer to the appendix.

STATEMENT	EXPLANATION
DIM C(50) S = 'THE GOOSE FLIES SOUTH' FOR I=1 TO LEN(STRING) C(I) = SEQ(S[I,1]) NEXT I	Encodes in vector C elements the decimal equivalents of individual characters of character string S.

Sample Usage of the SEQ Function.

9.101 SIN FUNCTION : SINE OF AN ANGLE

The SIN function generates the trigonmetric sine of an angle.

FORMAT:

SIN(expression)

To generate the sine of an angle expressed in degrees the SINE function is used. The given angle must be less than or equal to 14,073,748,834 and greater than or equal to -14,073,748,834.

Values which are less than 0 degrees, or greater than 360 degrees are adjusted to this range before generation.

(See: COSINE)

STATEMENT	EXPLANATION
YY = SIN(XX)	Assigns the sine of an angle of XX degrees to yYY.
PRINT SIN(1)	Prints "0.0174"
PRINT SIN(361)	Prints "0.0174"
PRINT SIN(2)	Prints "0.0349"
PRINT SIN(362)	Prints "0.0349"
PRINT SIN(45)	Prints "0.7071"
PRINT SIN(90)	Prints "1"

Sample usage of the SIN function.

9.102 SLEEP OR RQM STATEMENT : TIME ALLOCATION

The RQM or SLEEP statement terminates the executing program's current quantum (time-slice) The RQM or SLEEP statement may be used to effect program execution speed.

FORMAT:

RQM {seconds}	or	SLEEP {seconds}
RQM{"time.expression"}	or	SLEEP{"time.expression"}

The time-shared environment of the Pick system allows concurrent execution of several programs, with each program executing for a specific time period (called a time-slice or quantum) and then pausing while other programs continue execution. The RQM statement terminates the program's current time-slice. The RQM statement may be used in heavy compute loops to allow increased execution speed of other concurrently executing programs by giving up time. It may also be used to cause predetermined pauses (in seconds or until specified time) in program execution. The seconds parameter does not require quotes. The time expression (AM, PM or MILITARY) requires enclosure in quotes.

STATEMENTS	EXPLANATION
SLEEP 20	Sleep fo 20 seconds.
SLEEP "15:00"	Sleep until 3:00 PM.

```
* PROGRAM SEGMENT TO SOUND
* TERMINAL "BELL" FIVE TIMES.
BELL=CHAR(7)
FOR I=1 TO 5
PRINT BELL:                    RQM statement allows enough
RQM                            time for bell to be heard as
NEXT I                         discrete "beeps".
END
```

Sample usage of the SLEEP and RQM statements.

The SPACE function generates a string value containing a specified number of blank spaces.

FORMAT:

SPACE(length)

the SPACE function generates a string value containing the number of blank spaces specified by the length. For example:

```
PRINT SPACE(10):"HELLO"
```

This statement prints 10 blanks followed by the string "HELLO".

Conversely, the TRIM function is available to delete extraneous blanks.

(See: TRIM)

STATEMENT	EXPLANATION
B = 14 A = SPACE(B)	Assigns to variable A the string value containing 14 blank spaces.
DIM M(10) MAT M = SPACE(20)	Assigns a string consisting of 20 blanks to each of the 10 elements of array M.
S = SPACE(5) L = "SMITH" C = "," F = "JOHN" N = S:L:S:C:S:F	Assigns to variable N the concatenated string consisting of 5 blanks, the name SMITH, 5 blanks, a comma, 5 blanks, and the name JOHN.

Sample Usage of the SPACE Function.

The SQUARE ROOT function returns the positive square root of a positive number.

FORMAT:

SQRT(expression)

The SQUARE ROOT function returns the positive square root of any positive number (expression) that is greater than or equal to 0 and less than or equal 14,073,748,834.

STATEMENT	EXPLANATION
Y = SQRT(36)	Assign the value 6 to variable Y.
PRINT SQRT(1024)	Prints "32".
PRINT SQRT(1000)	Prints "31.6227"
PRINT SQRT(14073748834)	Prints "118632.832"

Sample Usage of the SQRT Function.

The STOP statement may appear anywhere in the program; it designates a logical termination of the program.

FORMAT:

```
STOP {errnum{,param, param, ...}}
```

Upon the execution of a STOP statement, the PICK/BASIC program will terminate.

The STOP statement may be placed anywhere within the PICK/BASIC program to indicate the end of one of several alternative paths of logic.

The STOP statement may optionally be followed by an error message name, and error message parameters separated by commas. The error message name is a reference to an item in the ERRMSG file. The parameters are variables or literals to be used within the error message format.

(See: ABORT)

```
      A=500 ; B=750 ; C=235 ; D=1300
      REVENUE=A+B  ;  COST=C+D
      PROFIT=REVENUE-COST
      IF PROFIT > 1 THEN GOTO 10
      PRINT "ZERO PROFIT OR LOSS"
      STOP <-------------------------- If this path taken,
 10   PRINT "POSITIVE PROFIT"           program will terminate
      END
```

Sample usage of the STOP Statement.

The STR function generates a string value containing a specified number of occurrences of a specified string.

FORMAT:

STR(expression,occurence#)

The STR function generates a string value containing the number of occurrences specified by the occurence# of the string specified by the expression. The following statement, for example, assigns a string value containing 12 asterisk characters to variable X:

```
X=STR('*',12)
```

As a further example, the following statement will cause the string value "ABCABCABC" to be printed:

```
PRINT STR('ABC',3) LEN(expression)
```

STATEMENT	EXPLANATION
VAR = STR("A",5)	Assigns to variable VAR the string value containing five A's.
A = 'BBB' B = STR("B",3) C = B CAT A	Assigns to variable C the string value containing six B's.
N = STR("?%?",4)	Assigns to variable N the string value containing 4 consecutive occurrences of the string "?%?".

Sample Usage of the STR Function.

9.107 SYSTEM FUNCTION : CALLING PRE-DEFINED SYSTEM VALUES

The SYSTEM function allows the user to obtain certain pre-defined values from the system. The value returned may either be an error status code (generated as a result of a previous BASIC statement), or a parameter such as the page-number of page-width.

FORMAT:

SYSTEM(expression)

The value of expression is in the range 0 through the maximum value as defined in table A. If the value of "expression" is outside the allowable range, the SYSTEM function will return a value as if the "expression" evaluated to zero (the error function).

If the expression used in the SYSTEM function is a zero, the function returns a value determined by the last executed BASIC statement that set an error condition. Examples of such BASIC statements are the tape commands such as READT, WRITET, etc. if the ELSE branch executes. SYSTEM(0), therefore, allows one to determine exactly what error has occurred when the program follows the ELSE branch of these statements. If the ELSE branch was not followed, the value returned by SYSTEM(0) is zero.

For example, the sequence of BASIC instructions:

```
READT TAPERECORD ELSE
    BEGIN CASE
      CASE SYSTEM(0) = 1; PRINT "ATTACH THE TAPE UNIT"; STOP
      CASE SYSTEM(0) = 2; PRINT "END OF FILE; DONE!"; STOP
    END CASE
END
```

will result in one of the messages being printed if there is either an EOF read from the tape, or if the tape unit was not attached to the line running the BASIC program.

The SYSTEM function, with non-zero values of the expression, returns parameters that have been set external to the BASIC program. See Table A.

Value of expression used in SYSTEM func.	Value returned
0	Error function value; see table B.
1	1 If PRINTER ON or (P) option used in RUN; 0 If data is being printed to the terminal.
2	Current page-size (page-width in columns).
3	Current page-depth (number of lines in page).
4	Number of lines remaining in current page.
5	Current line-counter (number of lines printed).
6	Current page-number.
7	One-character terminal-type code.
8	Current tape record length.
9	Current CPU millisecond count.
10	1 if current stack (STON) condition enabled. 0 if current stack inactive.
11	1 if LIST Function is active. 0 if LIST function is inactive.

Meaning of values usable in the SYSTEM function.

Previously executed BASIC statement.	Error code returned.	Meaning
READT, WRITET, WEOF or REWIND	1	Tape unit is not attached.
READT	2	EOF read from tape unit.
WRITET	3	Attempted to write null string.
	11	Attempted to write variable longer than tape record length.

Values returned by the error function, SYSTEM(0)

9.108 TAN FUNCTION : TANGENT OF AN ANGLE

The TAN function generates the trigonmetric tangent of an angle.

FORMAT:

TAN(expression)

To generate the tangent of an angle expressed in degrees the TAN function is used. The given angle must be less than or equal to 14,073,748,834 and greater than or equal to -14,073,748,834.

Values which are less than 0 degrees, or greater than 360 degrees are adjusted to this range before generation.

(See: COS and SIN)

STATEMENT	EXPLANATION
YY = TAN(XX)	Assigns the tangent of an angle of XX degrees to yYY.
PRINT TAN(1)	Prints "0.0174"
PRINT TAN(361)	Prints "0.0174"
PRINT TAN(2)	Prints "0.0349"
PRINT TAN(362)	Prints "0.0349"
PRINT TAN(45)	Prints "1"
PRINT TAN(90)	Prints "0"

Sample usage of the TAN function.

9.109 TIME() AND TIMEDATE() FUNCTIONS : TIME AND DATE CAPABILITY

The TIME() function returns the internal time of day. The TIMEDATE() function returns the current time and date in external format.

FORMAT:

```
TIME()

TIMEDATE()
```

The TIME() function returns the string value containing the internal time of day. The internal time is the number of seconds past midnight.

For example, at 4 minutes and 18 seconds after 5 P.M., the following statement would print ' 61458 '. (17:04:18 is 61458 seconds since midnight.)

```
PRINT TIME()
```

The TIMEDATE() function returns the string value containing the current time and date in the external format. This format is:

```
HH:MM:SS  DD MMM YYYY
          or
17:04:18  01 APR 1985
```

(See: DATE() function)

STATEMENT	EXPLANATION
A = TIME()	Assigns string value of current internal time to variable A.
IF TIME() > 1000 THEN GOTO 10	Branches to statement 10 if more than 1000 seconds have passed since midnight.
PRINT TIMEDATE()	Prints the current time and date in the external format.
WRITET TIME() ELSE STOP	Writes the string value of the current internal time onto a magnetic tape record.

Sample usage of the TIME() and TIMEDATE() functions.

9.110 TRIM FUNCTION : DELETING EXTRANEOUS SPACES

The TRIM function removes extraneous blank spaces from a specified string.

FORMAT:

TRIM(expression)

The TRIM function deletes preceding, trailing, and redundant blanks from the literal or variable expression. For example:

```
A='       GOOD MORNING,       MR. BRIGGS
A=TRIM(A)
PRINT A
```

The PRINT statement will print:

```
GOOD MORNING, MR. BRIGGS
```

Conversely, the SPACE function is available to generate blank spaces.

(See: SPACE)

STATEMENT	EXPLANATION
S = SPACE(5) L = "SMITH" C = "," F = "JOHN" N = S:L:S:C:S:F	Assigns to variable N the concatenated string consisting of 5 blanks, the name SMITH, 5 blanks, a comma, 5 blanks, and the name JOHN.
M = TRIM(N)	Assigns to variable M a string consisting of the name SMITH, 1 blank, a comma, one blank, and the name JOHN.

Sample Usage of the TRIM Function.

9.111 UNLOCK STATEMENT : CLEARING EXECUTION LOCKS

The UNLOCK statement provides a file and execution lock clearing capability for PICK/BASIC programs. The LOCK statement sets execution locks while the UNLOCK statement releases them.

FORMAT:

UNLOCK {expression}

The LOCK statement sets an execution lock so that when any other BASIC program attempts to set the same lock, then that program will either execute an alternate set of statements or will pause until the lock is released via an UNLOCK statement by the program which originally locked it.

The value of the expression specifies which execution lock is to be released (cleared). If the expression is omitted, then all execution locks which were previously set by the program will be released.

All execution locks set by a program will automatically be released upon termination of the program.

(See: LOCK)

STATEMENTS	EXPLANATION
UNLOCK 47	Resets execution lock 47.
UNLOCK	Resets all execution locks previously set by the program.
UNLOCK (5+A)*(B-2)	The value of the expression specifies which lock is released. execution lock is released.

Sample Usage of the UNLOCK Statement.

9.112 WEOF STATEMENT : POSITIONING TAPE

BASIC programs may specify Magnetic Tape positioning operations through the use of the WEOF (Write End-of-File Mark) statement.

FORMAT:

WEOF THEN/ELSE statements

The WEOF statement writes two EOF marks on the tape, then backspaces over the second one. This correctly positions the tape for subsequent WRITET operations.

(See: WRITET)

STATEMENT	EXPLANATION
WEOF ELSE STOP	Writes two EOF marks, then backspaces over the second one.

Sample usage of the WEOF statement.

The WRITE statement is used to update a file item.

FORMAT:

WRITE expression ON {file.variable,} itemname

The WRITE statement replaces the content of the item specified by itemname with the string value of the expression. The optional file.variable specifies the file variable; if it is used, the item will be replaced in the file previously assigned to that file.variable via an OPEN statement. If the file.variable is omitted, then the internal default variable is used (thus specifying the file most recently opened without a file variable). If the itemname specifies an item which does not exist, then a new item will be created.

The user should note that the PICK/BASIC program will abort with an appropriate error message if the specified file has not been opened prior to the execution of the WRITE statement.

(See: WRITEV and WRITET)

STATEMENT	EXPLANATION
WRITE "XXX" ON A, "ITEM5"	Replaces the current content of item ITEM5 (in the file opened and assigned to variable A) with string value "XXX".
A="123456789" B="X55" WRITE A ON FN1,B	Replaces the current content of item X55 (in the file opened and assigned to variable FN1) with string value "123456789".
WRITE 100*5 ON "EXP"	Replaces the current content of item EXP (in the file opened without a file variable) with string value "500".

Sample Usage of the WRITE Statement.

BASIC programs may specify Magnetic Tape output operations through the use of the WRITET (Write Tape Record) statement. The record length on the tape is as specified by the most recent T-ATT statement executed at the TCL level.

FORMAT:

WRITET expression THEN/ELSE statements

The WRITET statement writes a record onto the magnetic tape. The string value of the expression is written onto the next record of the tape.

If the tape unit has not been attached, or if the string value of the expression is the empty string (''), then the statement(s) following the ELSE will be executed.

(See: T-ATT and READT)

STATEMENT	EXPLANATION
FOR I=1 TO 5 WRITET A(I) ELSE STOP NEXT I	The values of array elements A(1) through A(5) are written onto 5 tape records. If one of the array elements has a value of '' (or if tape unit not attached), the program will terminate.

Sample Usage of the WRITET statement.

The WRITEU and WRITEVU statements have the letter "U" appended to them to imply update. The execution of these commands will not unlock the group locked by the program.

FORMAT:

WRITEU variable ON {file.variable,} itemname

WRITEVU variable ON {file.variable,} itemname,att#

These statements execute identically to the WRITE and WRITEV statements, with the following noted additional functionality.

(See: WRITE and WRITEV)

This version of these commands will not unlock the group locked by the program. This varient is used primarily for master file updates when several transactions are being processed and an update of the master item is made following each transaction update.

If the group is not locked when the WRITEU, WRITEVU or MATWRITEU statement is executed, the group will not be locked by the execution of the command.

STATEMENT	EXPLANATION
WRITEU CUST.NAME ON CUST.FILE,ID	Replaces the current contents of the item specified by variable ID (in the file opened and assigned to variable CUST.FILE) with with the contents of CUST.NAME. Does not unlock the group.
WRITEVU CUST.NAME ON CUST.FILE,ID,3	Replaces the third attribute of item ID (in the file opened and assigned to variable CUST.FILE) with the contents of variable CUST.NAME. Does not unlock the group.

Sample usage of WRITEU and WRITEVU statements.

9.116 WRITEV STATEMENT : UPDATING AN ATTRIBUTE

The WRITEV statement is used to write (update) a single attribute value to an item in a file.

FORMAT:

WRITEV expression ON {file.variable,} itemname,att#

Upon the execution of the WRITEV statement, the value of the expression becomes the attribute specified by att# (attribute number), in the item specified by the itemname and in the file previously assigned to the specified file.variable via an OPEN statement.

If the file.variable is omitted, then the internal default variable will be used (thus specifying the file most recently opened without a file.variable).

If a non-existent item name (or attribute number) is specified, then a new item (or attribute) will be created.

The WRITEV statement will also allow the attribute number (att#) to have a value of either zero or minus one, thus inserting data prior to the first attribute or following the last attribute.

When att# = 0, the expression is inserted at the begining of the item. All attributes in the item are shifted by 1 attribute and the expression becomes attribute 1.

When att# = -1, the expression is appended to the end of the item. The number of attributes in the item increase by 1 and all previously existing attributes are undisturbed.

The PICK/BASIC program will abort with an appropiate error message if the specified file has not been opened prior to the execution of the WRITEV Statement.

STATEMENT	EXPLANATION
Y="THIS IS A TEST" WRITEV Y ON X,"PROG",0	The string value "THIS IS A TEST" is inserted prior to the first attribute of item PROG in the file opened and assigned to variable X.
WRITEV "XYZ" ON "A7",4	Attribute 4 of item A7)in the file opened without a file variable) is replaced by string value "XYZ".

Sample usage of the WRITEV statements.

9.117 PICK/BASIC SYMBOLIC DEBUGGER : AN OVERVIEW

The PICK/BASIC Symbolic Debugger facilitates the debugging of new PICK/BASIC programs and the maintenance of existing PICK/BASIC programs.

When a PICK/BASIC program is compiled a symbol table item is automatically generated unless the suppress option (S) has been used. This table is used by the PICK/BASIC Debugger to reference symbolic variables during program execution.

The PICK/BASIC Debugger may be entered at execution time by 1) depressing the BREAK key or 2) using the 'D' (debug) option with the RUN verb. Once the PICK/BASIC Debugger has entered it will indicate the source code line number about to be executed and will prompt for commands with an asterisk (*) as opposed to the System Debugger prompt '!' or the TCL prompt.

The user now has at his disposal the following general capabilities: Controlled stepping through execution of program by way of single or multiple steps. Transferring control to a specified step (line number). Breaking (temporary halting) of execution on specified line number(s) or on the satisfaction of specified logical conditions. Displaying and/or changing any variable(s), including dimensioned variables. Tracing variables. Conditional entry to the System Debugger. Directing output (terminal/printer). Stack manipulation (displaying and/or popping the stack). Displaying of specified (or all) source code line(s).

The symbol table is embedded in the object code which is placed in the catalog space. The debugger has instant access to the symbol table, and requires the use of the 'Z' command only when access to the source code is required. Note that the user may suppress generation of the symbol table by using the (S) option when compiling programs.

A user requires SYS2 privileges to use the PICK/BASIC debugger. This prevents users from making unauthorized changes to data during reporting and data entry.

BASIC DEBUGGER FEATURE	RELATED COMMAND
1. Set breakpoint on logical condition where 'o' is logical operator <,>,=,#, 'v' is variable, 'c' is condition to be met, or 'n' is line number where preceded by B$o.	Bvoc{&voc} or B$on
2. Display breakpoint table	D
3. Escape to System Debugger	DEBUG or DE
4. Single/multiple step execution	En
5. End program execution and return to TCL	END
6. Proceed from breakpoint to specified line 'n'	G Gn
7. Remove all breakpoints specified breakpoint 'n'	K Kn
8. Display source code current line 'n' number of lines from current one number of lines from m-n all lines	L Ln Lm-n L*
9. Toggle output device (terminal & printer)	LP
10. Pass one breakpoint before stopping 'n' breakpoints	N Nn
11. Logoff	OFF
12. Inhibit output	P
13. Printer-close output to spooler	PC
14. Pop return stack Display return stack	R S
15. Switch turns trace table on/off Trace specified variable 'v'	T Tv
16. Remove all traces specified trace	U Un
17. Request symbol table	Z
18. Display current line number Print value of variable 'v' of element 'x' in array 'm' of element 'x,y' in matrix 'm' of entire array 'm' entire symbol table Set window remove window setting	$ /v /m(x) /m(x,y) /m /* [x,y] [

9.118 USING THE PICK/BASIC DEBUGGER : AN EXAMPLE

The following is a step-by-step introduction to the use of the PICK/BASIC DEBUGGER for inexperienced users. This will demonstrate only a few of the commands as it is merely intended to give the user an introductory "feeling" for the use of the PICK/BASIC DEBUGGER.

A sample program "SAMPLE" is shown below, followed by steps a user might take to debug it.

SAMPLE

```
001 DIM ARRAY(10) ; * ARRAY HAS 10 SLOTS
002 FOR I = 1 TO 20 ; * BUG: LOOP SPECIFIES 20 PASSES, ARRAY HAS ONLY 10
003   ARRAY(I) = I ; * EACH SLOT WILL BE FILLED WITH A CONSECUTIVE #
004 NEXT I
005 PRINT ARRAY(I)
006 END
```

"SAMPLE" compiles without any errors detected. Once it is run however, it aborts with the error message "ARRAY SUBSCRIPT OUT OF RANGE" and traps to the PICK/BASIC DEBUGGER. Supposing that the user cannot find the error, the following steps could be taken for detecting the error using the PICK/BASIC DEBUGGER.

1. The user enters the command "Z" to the DEBUGGER prompt character "*". The DEBUGGER responds with "PROG NAME?", the user enters the program name. This allows the DEBUGGER access to the symbol table created during compilation. Alternatively, if the user uses the debug option "(D)" during run time, access to the symbol table is already established, and use of the "Z" command is unnecessary.

2. To find out how far in the loop the program progressed, the user looks at the variable "I" by entering "/I". The DEBUGGER responds with "11 =", at which the user may change the value of "I" if desired. The user may then want to look at all of the values in the array by entering "/ARRAY". The DEBUGGER responds with "ARRAY(1)=1=", the user depresses return and the DEBUGGER continues with the next "array slot" (i.e., "ARRAY(2)=2=" etc.). Once "ARRAY(10)=10=" has been reached the user presses return and the DEBUGGER returns with the "*" prompt, the user knows that the array has only 10 slots and the loop calls for 20 -- thus he finds the error. The user may then end the "session" with PICK/BASIC DEBUGGER by entering "END" and repair the bug.

A summary of this interaction is given in Figure A on facing page. For purposes of clarity, whatever is entered by the user is shown enclosed in square brackets "[]". These are not part of the commands; they are to distinguish user entry from DEBUGGER response.

NOTE: The square brackets surround user input to distinguish it from PICK/BASIC Debugger responses. They are not part of the commands!

```
     LINE 3 (B17) ARRAY SUBSCRIPT OUT OF RANGE
     *I3
     *[Z] PROG NAME?[SAMPLE]
     *[/I]  [CR] 11=
     *[/ARRAY]  [CR] ARRAY(1)=1=  [CR]
       ARRAY(2)=2=  [CR]
       ARRAY(3)=3-  [CR]
       ARRAY(4)=4=  [CR]
       ARRAY(5)=5=  [CR]
            .
            .
            .
       ARRAY(10)=10=  [CR]
     *[CR]
     *[END]
```

Sample Session with the PICK/BASIC Debugger.

NOTE: A carriage return will return control to the BASIC DEBUGGER prompted by "*" whereas a line-feed will return control to program execution until a breakpoint, an error or the end of the program is met.

The trace table is used for the automatic printout of a specified variable or variables after a break has occured.

Up to six trace values may be entered in the table. Either the symbolic name, or a line number and variable number may be used to reference the variable. In addition, all the variables in the last statement executed may be printed out. The trace table may be alternately turned on and off by use of the "T" return command.

Examples of use of the trace table are shown below:

Tname	The value of the variable name will be printed out at each breakpoint.
T%10,3	The value of the third variable in line number 10 will be printed out at each breakpoint. If line number 10 contains the statement "A=B+C+D" the value of "C" will be printed.

To delete a variable from the trace table use the "U" command followed by the trace variable to be deleted. For example, to delete the variable name from the table type in "Uname". "U" return deletes the entire trace table.

If a program calls an external subroutine, and the BASIC/DEBUGGER has been entered previously, a complete symbol table will be set up for the external subroutine. the table will have 4 break-points and 6 variable traces available, as well as pointers to program source and object, which may be set up by the Z command. break points set up for a subroutine are independent from break points set up in the main program or other subroutines; however, the execution counters (E and N,) are global.

the use of multiple symbol tables allows the programmer to set up different break points and/or variable traces for different subroutines.

Additional commands to set (B)reakpoints, (D)isplay and (K)ill breakpoints.

FORMAT:

```
B[variable-name][operator][expression]{ & another condition}
B$[operator][line-number]{ & another condition}
```

Where 'variable-name' is a simple variable or an explicitly stated array element and 'expression' is a variable, constant, or array element. If the variable does not exist or if the wrong Symbol Table is assigned, the message "SYM NOT FND" will be printed. String constants must be enclosed in quotes using the same rules that apply to PICK/BASIC literals. The Breakpoint Table may contain up to four conditions that when satisfied, will cause a break in execution. Logical expressions are used to set the break conditions. The logical operators used are:

<	less than
>	greater than
=	equal to
#	not equal to
&	is used as a logical connector between conditions.
$	is a special symbol used to indicate that a line number is specified rather than a variable name.

A plus sign will be printed next to the command if it is accepted. When the condition is met, an execution break will occur and the Debugger will halt execution of the program and print *Bn 1 where 'n' is one of the 4 Breakpoint Table entries and '1' is the program line number that caused the break.

FORMAT: D

The 'D' command will display the Trace and Breakpoint Tables.

FORMAT:

Kn	Deletes 'n'th breakpoint. 'n' range 1 to 4.
K<return>	Deletes all breakpoint conditions.

The 'K' command is used to delete breakpoint conditions from the table. A minus sign will be printed next to the command to indicate that an entry has been removed.

COMMAND	EXPLANATION
BX<42	Breaks when X is less than 42.
BADDRESS=''	Breaks when ADDRESS is null.
BDATE=INV.DATE&$=22	Breaks when variable DATE is equal to variable INV.DATE and if the line number is 22.
K2	Kills the second breakpoint condition.
BPRICE(3)=24.98	Breaks when the third element of the array PRICE is equal to 24.98. Only individual array elements may be specified.
D	Displays the Trace and Breakpoint Tables.
K	Kills all breakpoint conditions.

Examples of B, D, and K Commands.

9.121 E(XECUTE), G(O) AND N(O or BYPASS) COMMANDS : DEBUGGER EXECUTION

The commands "E", "G", and "N" in conjunction with the breakpoint table control the execution of the program under debug control.

The "E" command will allow the execution of a specified number of lines before returning control to the user. The number of statements to be executed is selected by putting a numeric value after "E". For example "E3" will executed three line statements before returning control to the user. "E" return will turn off the "E" command.

COMMAND	RESULT
E10	Ten line statements will be executed before control returns to the user.
E	Execution continues until interuption by the user, by a breakpoint or until program ends.

The "N" command will allow the user to bypass any number of breakpoints before control is passed back to the user, however, the trace table variables will be printed at each breakpoint. "N0" equals 'pass one breakpoint', "N1" equals 'pass two breakpoints', etc. and "N" return will set "N" to "N0".

COMMAND	RESULT
N3	Four breakpoints are passed, although the trace table values, if present, are printed out at each breakpoint. Control is then returned to the user.

The "G" command followed by a line number will allow control to be passed to the line number indicated. The "G" return command will cause program to execute the next command from the current line number and it will return control depending on the breakpoint setup.

COMMAND	RESULT
G153	Control passes to line number 153 and thereafter to user.
G	Control passes to next program line and thereafter to user.

9.122 SLASH '/' COMMAND : DISPLAYING AND CHANGING VARIABLES

Variables and arrays can be displayed and changed in either decimal or string formats.

To display a variable use the command '/v' where 'v' is a variable. For example to display the value of the variable name select '/name'. The DEBUGGER will respond with the string in the name field and an equal sign. If the variable is not to be changed press return. If the variable is to be changed put in the new value of the variable desired and press return. To display a complete array just place the name of the array after the slash. To display one value in the array use the form '/M(x)' or '/M(x,y)' where 'x' and 'y' are points in the array. The array point may then be changed in the same way as for a single variable.

A window may be placed after any variable selection by following the variable with a ';' and the length of the window. For example, to limit the variable name to eight characters the command '/name;8' would be used. Numeric variables will ignore any window commands.

The symbolic name of the variable may be replaced with the form '%x,y' where 'x' is the line number and 'y' is the nth variable in that line in the same way as the breakpoint table. Examples of displaying and changing variables follows:

/CITY IRVINE=	The variable 'city' is displayed but not changed.
/STATE NY=CA	The variable 'state' is displayed as 'NY' and changed to 'CA'
/FIELD(5) 10=	The fifth point in array FIELD is displayed as 10 and not changed.
/*	All the symbols in the symbol table are displayed.

Additional PICK/BASIC Debugger commands : PC, P, LP, $, L, Z, [], DEBUG and END.

I/O CONTROL

The "P" command inhibits all PICK/BASIC program output so that the user may look only at the DEBUGGER output. "P" return alternately turns "P" on and off.

The "LP" command forces all output to the line printer which can be used for a fast trace or hard copy of a trace. "LP" return alternately turns the line printer command on and off.

The "PC" command is the same as the PICK/BASIC printer close command. All data that is waiting to be sent to the printer is output at this time.

SOURCE CODE DISPLAY

The "$" command will print the next line number to be executed.

The "L" command will display sorce code lines. "L" will display the current line of source. "Ln" will display line 'n'. "Lm-n" will display lines 'm-n'. "L*" will display the entire source program.

SYMBOL TABLE

The "Z" command will allow the operator to specify a symbol table. After the operator enters the program name, that symbol table if present, will be enabled. The program name may be entered as "item-name", "file-name item-name", or "file-name,dataname item-name".

STRING WINDOWS

The string window command "[n,m]" will cause the output of all variables to be limited to the substring selected. An example of the command follows:

```
X=1234567890
```

[3,2] Sets the window for the third character position with a string length of two. Any printout of x will be 34.

Setting the window length to zero will turn the string window command off. "[Carriage-return" will have the same result.

ESCAPE TO SYSTEM DEBUGGER

The "DEBUG" command will pass control to the System DEBUGGER.

TERMINATION

The "END" command will terminate the PICK/BASIC and DEBUG programs and return control to TCL.

This topic presents a number of general coding techniques which the programmer should keep in mind when writing PICK/BASIC programs.

The PICK system uses standard attribute and value delimiters. These should be defined once in the initialization portion of the program, and then referenced by their variable name. for example:

```
AM = CHAR(254)      Attribute Mark
VM = CHAR(253)      Value Mark
SVM = CHAR(252)     Secondary Value Mark
```

Cursor positioning should be controlled by the following PRINT Statements using the @ Functions and the Character Functions:

```
ERASE SCREEN = PRINT @(-1)                  HOME = PRINT @(-2)
CLEAR TO END OF SCREEN = PRINT @(-3)        CLEAR TO END OF LINE = PRINT @(-4)
START BLINK = PRINT @(-5)                   STOP BLINK = PRINT@(-6)
START PROTECT = PRINT @(-7)                 STOP PROTECT = PRINT @(-8)
BACKSPACE = PRINT @(-9)                     UP 1 LINE = PRINT @(-10)
DOWN 1 LINE = CHAR(10)                      RIGHT 1 CHARACTER = CHAR(6)
BELL = CHAR(7)
```

The OPEN statement is very time consuming and should be executed as few times as possible. All files should be opened to file variables at the beginning of the program; access to the files can then be performed by referencing the file variables.

The size of programs can be reduced, with a corresponding increase in overall system performance, by reducing the amount of literal storage. For example:

```
200 PRINT 'RESULT IS ':A+B
210 PRINT 'RESULT IS ':A-B
220 PRINT 'RESULT IS ':A*B
230 PRINT 'RESULT IS ':A/B
```

These statements should have been written as follows:

```
    MSG = 'RESULT IS'
    .
    .
    .
200 PRINT MSG:A+B
210 PRINT MSG:A-B
220 PRINT MSG:A*B
230 PRINT MSG:A/B
```



```
*****************************************************************
*  THIS PROGRAM FORMATS A PICK/BASIC PROGRAM TO                 *
*  DISPLAY BLOCK STRUCTURING BY INDENTING LINES.                *
*****************************************************************
*---- DEFINITIONS
10    SP = 6           ;* LEFT MARGIN COLUMN NUMBER
      ID = 3           ;* NUMBER OF SPACES TO INDENT
*---- INITIALIZATION
      SPX = SP
      LINE.NO = 0
*---- INPUT FILE NAME AND  PROGRAM NAME
      PRINT
      PRINT
      PRINT 'DATA/BASIC FILE NAME    - ':; INPUT FILE
      IF FILE = '' THEN STOP
      OPEN '',FILE ELSE PRINT 'CANNOT OPEN FILE - ': FILE;  GOTO 10
      PRINT 'DATA/BASIC PROGRAM NAME - ':; INPUT NAME
      IF NAME = '' THEN GOTO 10
      NEWITEM = ''
      READ ITEM FROM NAME ELSE
         PRINT 'CANNOT FIND THAT PROGRAM'
         GOTO 10
      END
*---- GET NEW LINE, IF NONE - THEN DONE
100   LINE.NO = LINE.NO + 1
      LINE = EXTRACT(ITEM,LINE.NO,0,0)
      IF LINE = '' THEN
         WRITE NEWITEM ON NAME
         PRINT; PRINT; PRINT '--DONE--'; GOTO 10
      END
      LABEL = ''
*---- STRIP OFF LEADING/TRAILING SPACES
200   IF LINE[1,1] = ' ' THEN LINE = LINE[2,32767]; GOTO 200
210   IF LINE[LEN(LINE),1] = ' ' THEN LINE = LINE[1,LEN(LINE)-1]; GOTO 210
*---- LOOK FOR A COMMENT ('*', '!', OR 'REM')
      IF LINE[1,1] = '*' THEN GOTO 1500
      IF LINE[1,1] = '!' THEN GOTO 1500
      IF LINE[1,3] = 'REM' THEN GOTO 1500
*---- LOOK FOR 'FOR'
      IF LINE[1,4]='FOR ' AND INDEX(LINE,'NEX ',1)>0 THEN GOTO 2000
      IF LINE[1,4]='FOR ' AND INDEX(LINE,'NEXT ',1)=0 THEN GOTO 1000
*---- LOOK FOR 'END'
      IF LINE = 'END' THEN GOTO 1100
      IF LINE[1,4] = 'END ' THEN
         IF LINE[LEN(LINE)-4,5] = ' ELSE' THEN GOTO 1200
      END
*---- LOOK FOR 'NEXT'
      IF LINE[1,5] = 'NEXT ' THEN GOTO 1100
*---- EXTRACT LEADING NUMERIC LABEL
      IF LINE[1,1] MATCHES '1N' THEN
         L = 2
```



```
300   IF LINE[L,1] MATCHES '1N' THEN L=L+1; GOTO 300
      LABEL = LINE[1,L-1]
      LINE = LINE[L,32767]
      GOTO 200
      END
*---- LOOK FOR LINE ENDING IN ' ELSE' OR ' THEN'  ('IF' OR 'READ')
      X = LINE[LEN(LINE)-4,5]
      IF X = ' THEN' THEN GOTO 1000
      IF X = ' ELSE' THEN GOTO 1000
*---- THIS IS JUST ANOTHER LINE, THEREFORE NO CHANGE
      GOTO 2000
*---- INDENT ON SUBSEQUENT LINES
1000  SP = SP + ID
GOTO 2000
*---- OUTDENT ON THIS AND SUBSEQUENT LINES
1100  SP = SP - ID
*---- OUTDENT THIS LINE ONLY
1200  SPX = SPX - ID
      GOTO 2000
*---- PRINT WITH NO INDENTATION
1500  SPX = 0
*---- WRITE NEW LINE
2000  NEW.LINE = LABEL : STR(' ',SPX-LEN(LABEL)) : LINE
      PRINT NEW.LINE
      NEWITEM = REPLACE(NEWITEM,LINE.NO,0,0,NEW.LINE)
      SPX = SP
      GOTO 100
   END
```

9.129 PROGRAMMING EXAMPLES: LOT-UPDATE

```
*********************************************************
*  THIS PROGRAM UPDATES DATA ON LOTS IN A HOUSING TRACT. *
*  ITEM-ID'S IN "LOT" FILE ARE TRACT.NAME*LOT.NUMBER     *
*********************************************************
100* INITIALIZATION
     PROMPT '='
     CLEAR
     DIM DESC(30),TYPE(30)
     OPEN 'DICT','LOT' ELSE
        PRINT "CAN'T OPEN DICT LOT"
        STOP
     END
*
200* GET DESCRIPTIONS, CONVERSIONS
     FOR I = 1 TO 30
        READ DICT.ITEM FROM I ELSE
           PRINT "DICTIONARY ITEM '":I:"' NOT FOUND"
           GOTO 250
        END
        D = EXTRACT(DIC.ITEM,3,0,0)          ;* S/NAME--DESCRIPTION
        IF D # '' THEN DESC(I) = D:STR('.',15-LEN(D))
        IF C[1,2] = 'MD' THEN
           TYPE(I) + 'NUM'
           GOTO 250
        END
        IF C[1,1] = '0' THEN TYPE(I) = 'DATE'
250*
     NEXT I
*
*
     OPEN '','LOT' ELSE
        PRINT "CAN'T OPEN LOT FILE."
        STOP
     END
*
300* GET THE TRACT NAME
     PRINT
     PRINT "TRACT NAME.....":
     INPUT TRACT
     IF TRACT = 'STOP' OR TRACT = 'END' THEN STOP
     IF TRACT = '' THEN GOTO 300
     READ INFO FROM TRACT ELSE
        PRINT "TRACT '":TRACT:"' OT ON FILE"
        GOTO 300
     END
*
400* GET A VALID LOT NUMBER
     PRINT
     PRINT "LOT NUMBER.....":
     INPUT NUMBER
     IF NUMBER = '' THEN GOTO 400
     IF NUMBER = 'END' OR NUMBER = 'STOP' THEN GOTO 300
     IF NUM(NUMBER) = 0 THEN
        PRINT "MUST BE A NUMBER"
        GOTO 400
```

```
      END
      NUMBER = TRACT:'*':NUMBER
      READ ITEM FROM NUMBER ELSE
         ITEM = ''
         PRINT "NEW LOT"
      END
*
450*
      NOT.SOLD = 0
      FOR I = 1 TO 30
            GOSUB 1000                    ;* UPDATES THE I'TH ATTRIBUTE
         IF I = 10 THEN
            IF EXTRACT(ITEM,10,0,0) = '' THEN
               NOT.SOLD = 1
               I = 19
            END
         END
*
         IF I = 21 THEN
            IF NOT.SOLD THEN GOTO 500
         END
      NEXT I
*
      VERITY DATA & STORE
      PRINT
      PRINT"         OK      ":
      INPUT OK
      IF OK = '' THEN
            WRITE ITEM ON NUMBER
            GOTO 400
      END
      IF OK = 'L' THEN
         PRINT
         FOR L = 1 TO 30
            ATT = EXTRACT(ITEM,I,0,0)
            IF ATT = '' THEN GOTO 550
            PRINT DESC(L):
            IF TYPE(L) = 'DATE' AND NUM(DATE) THEN ATT = OCONV(ATT,'DO')
            IF TYPE(L) = 'NUM' AND NUM(ATT) THEN ATT = 0.01 * ATT
            PRINT ATT 'R################'
550*
         NEXT L
         GOTO 500
      END
      GOTO 400
*
1000* UPDATE'S THE I'TH ATTRIBUTE OF "ITEM"
      IF DESC(I) = '' THEN RETURN          ;* NOT NEEDED OR NOT FOUND
      PRINT DESC(I):
      CURRENT = EXTRACT(ITEM,I,0,0)
*
      IF TYPE(I) = 'NUM' THEN
1100* NEED A NUMBER (AMOUNT)
         PRINT CURRENT*.01  'R###############':
         INPUT RESPONSE
         IF RESPONSE = '' THEN RETURN      ;* JUST LOOKING
         IF RESPONSE = '' THEN
            ITEM = REPLACE(ITEM,I,0,0,'')
```



```
            RETURN                              ;* DELETE THIS ATT.
         END
         IF NUM(RESPONSE) = 0 THEN
            PRINT "MUST BE A NUMBER"
            GOTO 1100
         END
*
         ITEM = REPLACE(ITEM,I,0,0,RESPONSE*100)
         RETURN
      END
*
      IF TYPE(I) = 'DATE' THEN
1200* NEED A DATE
         PRINT OCONV(CURRENT,'DO') 'R################':
         INPUT RESPONSE
         IF RESPONSE = '' THEN RETURN       ;* JUST LOOKING
         IF RESPONSE = 'T' THEN
            DATE = DATE()
            GOTO 1250
         END
         IF RESPONSE = '' THEN
            ITEM = REPLACE(ITEM,I,0,0,'')   ;* DELETE THIS ATT.
            RETURN
         END
         DATE = ICONV(RESPONSE,'D')
         IF DATE = '' THEN
            PRINT "USE DATE FORMAT 'MONTH/DAY/YEAR'"
            GOTO 1200
         END
1250*
         ITEM = REPLACE(ITEM,I,0,0,DATE)
         RETURN
      END
1300* NO NECESSARY FORMATS
      PRINT CURRENT 'R################':
      INPUT RESPONSE
      IF RESPONSE = '' THEN RETURN
      IF RESPONSE = '' THEN RESPONSE = ''
      ITEM = REPLACE(ITEM                I,0,0,RESPONSE)
      RETURN
END
```

SUMMARY OF PICK/BASIC STATEMENTS

This appendix presents the general form for each of the PICK/BASIC Statements. The statements are listed in alphabetical order.

STATEMENTS

```
ABORT{errnum{,param,param,...}}

BREAK ON/OFF

CALL @name(argument list)

CALL name(argument list)

CASE --- BEGIN CASE

             CASE expression
             stmts
             CASE expression
             stmts
             .
             .
         END CASE

CHAIN "any TCL command"

CLEAR

CLEARFILE {file.variable}

COM{MON} variable {,variable}

DATA expression{,expression...}

DELETE {file.variable,} itemname

DIM variable(dimensions) {,variable(dimensions)}

ECHO ON/OFF

END

EQU{ATE}  variable TO equate-variable {, ...}

FOOTING "text 'options' {text 'options'}"

FOR...NEXT---FOR variable = exp TO exp {STEP exp}{WHILE/UNTIL exp}

             NEXT variable

GOSUB statement-label
```



```
GO{TO} statement-label
HEADING "text 'options' {text 'options'}"
IF expression THEN stmts {ELSE stmts}
INPUT variable {:}
INPUT @(column,row) : variable {'mask'}
INPUTERR expression
INPUTTRAP'xx' GOTOn,n,n...
INPUTTRAP 'xx' GOSUB n,n,n...
INPUTNULL x
LOCATE('String',Item{,Att#{,Val#}};index#{;sequence}) THEN/ELSE stmts
LOCK expression {THEN/ELSE stmts}
LOOP {stmts} WHILE/UNTIL expression DO {stmts} REPEAT
MAT variable
MAT array.variable = expression
MAT array.variable = MAT array.variable
MATREAD array.variable FROM {file.variable,} itemname THEN/ELSE stmts
MATREADU array.variable FROM {file.variable,} expression THEN/ELSE stmts
MATWRITE array.variable ON {file.variable,} expression
MATWRITEU array.variable ON {file.variable,} expression
NEXT variable
NULL
ON expression GOTO/GOSUB statement-label, statement-label
OPEN {"DICT",} "filename" {TO file.variable} THEN/ELSE stmts
PAGE {expression}
PRECISION n
PRINT {ON expression} print-list
PRINTER ON/OFF
PRINTER CLOSE
```



```
PROMPT expression
READ variable FROM {file.variable,} itemname THEN/ELSE stmts          READ
READNEXT variable {,vmc}{FROM select.variable} THEN/ELSE stmts
READT variable THEN/ELSE/ stmts
READU variable FROM {file.variable,} itemname THEN/ELSE stmts
READV variable FROM {file.variable,} itemname,att# THEN/ELSE stmts
READVU variable FROM {file.variable,} itemname,att# THEN/ELSE stmts
RELEASE {{file.variable,} expression}
REM   or    *     or   !
RETURN
RETURN TO statement-label
REWIND THEN/ELSE stmts
RQM {seconds or "time"}
SELECT {file.variable}{TO select.variable}
SLEEP {seconds or "time"}
STOP {errnum{param,param,...}}
SUBROUTINE name (argument list)
UNLOCK {expression}
WEOF THEN/ELSE {expression}
WRITE expression ON {file.variable,} itemname
WRITEU variable ON {file.variable,} itemname
WRITET expression THEN/ELSE stmts
WRITEV expression ON {file.variable,} itemname,att#
WRITEVU expression ON {file.variable,} itemname,att#
```

BASIC INTRINSIC FUNCTION SUMMARY

This appendix presents the general form for each of the PICK/BASIC Intrinsic Functions. The functions are listed in alphabetical order. page referenced.

FUNCTION

@(column{,row})

ABS(expression)

ALPHA (expression)

ASCII(expression)

CHAR(expression)

COL1()

COL2()

COS (expression)

COUNT (string,substring)

DATE()

DCOUNT (string,substring)

DELETE(da.expression,att#{,value#{,sub-value#}})

EBCDIC(expression)

EXP(expression)

EXTRACT(da.expression,att#{,value#{,sub-value#}})

FIELD(expression,delimiter,occurence#)

ICONV(expression,conversion)

INDEX(string,sub-string,occurence#)

INSERT(da.expression,att#{,value#{,sub-value#,}}{;}new.expression)

INT(expression)

LEN(expression)

LN


```
MOD(numerator,denominator)

NOT(expression)

NUM(expression)

OCONV(expression,conversion)

PWR(expression,power)

REM(numerator,denominator)

REPLACE(da.expression,att#{,value#{,sub-value#,}}{;}new.expression)

RND(expression)

SEQ(expression)

SIN (expression)

SPACE(length)

SQRT (expression)

STR(expression,occurence#)

TAN (expression)

TIME()

TIMEDATE()

TRIM(expression)
```

BASIC COMPILER ERROR MESSAGES

This appendix presents a list of the error messages which may occur as a result of compiling a PICK/BASIC program.

ERROR NO.	ERROR MESSAGE	CAUSE
B0	PROGRAM 'xx' COMPILED. n FRAMES USED.	PICK/BASIC program compiled with no compilation errors. This is not an error; it simply informs that compilation is completed.
B100	COMPILATION ABORTED; NO OBJECT CODE PRODUCED	Compilation errors present.
B101	MISSING "END", NEXT", "WHILE", "UNTIL", "REPEAT", OR "ELSE"; COMPILATION ABORTED, NO OBJECT CODE PRODUCED	Compilation error present.
B102	BAD STATEMENT	Unrecognizable statement.
B103	LABEL "C" IS MISSING	Label indicated by GOTO or GOSUB was not found.
B104	LABEL "C" IS DOUBLY DEFINED	More than one statement was found beginning with the same label.
B105	"C" HAS NOT BEEN DIMENSIONED	Subscripted variable was not dimensioned.
B106	"C" HAS BEEN DIMENSIONED AND USED WITHOUT SUBSCRIPTS	Dimensioned variable used without subscripts.
B107	"ELSE" CLAUSE MISSING	ELSE clause is missing.
B108	"NEXT" STATEMENT MISSING	NEXT statement is missing in FOR-NEXT loop.
B109	VARIABLE MISSING IN "NEXT" STATMENT	Iteration variable is missing in NEXT statement.
B110	"END" STATEMENT MISSING	END statement is missing in multi-line IF statement.
B111	"UNTIL" OR "WHILE" MISSING IN "LOOP" STATEMENT	UNTIL or WHILE clause is missing in a LOOP statement.

B112	"REPEAT" MISSING IN "LOOP" STATEMENT	REPEAT is missing in a LOOP statement.
B113	COLUMN B TERMINATOR MISSING	Garbage following a legal statement or quote missing.
B114	MAXIMUM NUMBER OF VARIABLES EXCEEDED	Using the default descriptor size of 10, the maximum number of variables (including array elements) is 3274.
B115	LABEL 'C' IS USED BEFORE THE EQUATE STMT	The equate-variable is referenced before it has been defined.
B116	LABEL 'C' IS USED BEFORE THE COMMON STMT	A common variable has been referenced before it is put in common.
B117	LABEL 'C' IS MISSING A SUBSCRIPT LIST	An array is referenced without a subscript list.
B118	LABEL 'C' IS THE OBJECT OF AN EQUATE STMT AND IS MISSING	
B119	WARNING - PRECISION VALUE OUT OF RANGE - IGNORED!	
B120	WARNING - MULTIPLE PRECISION STATEMENTS - IGNORED!	
B121	LABEL 'C' IS A CONSTANT AND CAN NOT BE WRITTEN INTO	
B122	LABEL 'C' IS IMPROPER TYPE	

BASIC RUN-TIME ERROR MESSAGES

This appendix presents a list of the error messages which may occur as a result of executing a PICK/BASIC program. Warning messages indicate that illegal conditions have been smoothed over (by making an appropriate assumption), and do not result in program termination. Fatal error messages result in program termination.

WARNING MESSAGES

Error No.	Error Message	Cause
B10	VARIABLE HAS NOT BEEN ASSIGNED A VALUE; ZERO USED!	An unassigned variable was referenced. (A value of 0 is assumed.)
B11	TAPE RECORD TRUNCATED TO TAPE RECORD LENGTH!	An attempt was made to write more onto a tape record than the tape record length. (The record is truncated to tape record length.)
B13	NULL CONVERSION CODE IS ILLEGAL; NO CONVERSION DONE!	A string variable that should have a value is actually null.
B16	NON-NUMERIC DATA WHEN NUERIC REQUIRED; ZERO USED!	A non-numeric string was encountered when a number was required. (A value of 0 is assumed.)
B19	ILLEGAL PATTERN	Illegal pattern used with MATCH or MATCHES operator.
B20	COL1 OR COL2 USED PRIOR TO EXECUTING A FIELD STMT; ZERO USED!	COL1 or COL2 function used before FIELD function used. (A value of 0 is assumed.)
B24	DIVIDE BY ZERO ILLEGAL; ZERO USED!	Division by zero attempted. (A value of 0 is assumed.)
B209	FILE IS UPDATE PROTECTED	
B210	FILE IS ACCESS PROTECTED	

Error No.	Error Message	Cause
B12	FILE HAS NOT BEEN OPENED	File indicated in I/O statement has not been opened via an OPEN statement.
B14	BAD STACK DESCRIPTOR	This error message is generated if the the lengths of the input-lists or lengths of the input-lists or output-lists in the CALL and SUBROUTINE statements are different, if an attempt is made to execute an external subroutine as a main program or if a file variable is used as an operand.
B15	ILLEGAL OPCODE: C	Object code for this item is not legal.
B17	ARRAY SUBSCRIPT OUT-OF-RANGE	Array subscript is less than or equal to zero or exceeds the row or column number indicated by a DIM statement.
B18	ATTRIBUTE LESS THAN -1 IS ILLEGAL	Attribute less than on specified in READV or WRITEV statement.
B25	PROGRAM "C" HAS NOT BEEN CATALOGED	The specified external subroutine must be cataloged before appearing in a CALL statement.
B27	RETURN EXECUTED WITH NO GOSUB	RETURN statement executed prior to GOSUB.
B28	NOT ENOUGH WORK SPACE	Not enough work space assigned at LOGON to run program.
B30	ARRAY SIZE MISMATCH	Array sizes in MAT Copy statement, or in CALL and SUBROUTINE statements, do not match.
B31	STACK OVERFLOW	The program has attempted to call too many nested subroutines.
B32	PAGE HEADING EXCEEDS MAXIUM OF 1400 CHARACTERS	Page heading is too long.
B33	PRECISION DECLARED IN SUBPROGRAM 'C' IS DIFFERENT FROM THAT DECLARED	Precision must be the same between calling programs and subroutines.
B34	FILE VARIABLE USED WHERE STRING EXPRESSION EXPECTED	
B41	LOCK NUMBER IS GREATER THAN 47	Maxium of locks available is 47.

9.134 LIST OF ASCII CODES

This appendix presents a list of ASCII codes used in the PICK system.

DECIMAL	HEX	CHARACTER	SPECIAL USE IN PICK
0	0	NUL	Null prompt character
1	1	SOH	Cursor home on CRT Terminal
2	2	STX	
3	3	ETX	
4	4	EOT	
5	5	ENQ	
6	6	ACK	Cursor forward on CRT Terminal
7	7	BEL	Bell on CRT Terminal
8	8	BS	
9	9	HT	
10	A	LF	Cursor down on CRT Terminal
11	B	VT	Vertical address on CRT Terminal
12	C	FF	Screen erase on CRT Terminal
13	D	CR	Carriage return on CRT Terminal
14	E	SO	
15	F	SI	
16	10	DLE	Horizontal address on CRT Terminal
17	11	DC1	
18	12	DC2	
19	13	DC3	
20	14	DC4	
21	15	NAK	Cursor back on CRT Terminal
22	16	SYN	
23	17	ETB	
24	18	CAN	
25	19	EM	
26	1A	SUB	Cursor up on CRT Terminal
27	1B	ESC	
28	1C	FS	
29	1D	GS	
30	1E	RS	
31	1F	US	
32	20	SPACE	
33	21	!	
34	22	"	
35	23	#	
36	24	$	
37	25	%	
38	26	&	
39	27	'	
40	28	(	
41	29	)	
42	2A	*	
43	2B	+	
44	2C	'	
45	2D	-	
46	2E	.	
47	2F	/	
48	30	0	

DECIMAL	HEX	CHARACTER	SPECIAL USE IN PICK
49	31	1	
50	32	2	
51	33	3	
52	34	4	
53	35	5	
54	36	6	
55	37	7	
56	38	8	
57	39	9	
58	3A	:	
59	3B	;	
60	3C	<	
61	3D	=	
62	3E	>	
63	3F	?	
64	40	@	
65	41	A	
66	42	B	
67	43	C	
68	44	D	
69	45	E	
70	46	F	
71	47	G	
72	48	H	
73	49	I	
74	4A	J	
75	4B	K	
76	4C	L	
77	4D	M	
78	4E	N	
79	4F	O	
80	50	P	
81	51	Q	
82	52	R	
83	53	S	
84	54	T	
85	55	U	
86	56	V	
87	57	W	
88	58	X	
89	59	Y	
90	5A	Z	
91	5B	[	
92	5C	/	
93	5D	[	
94	5E		
95	5F		
123	7B		
124	7C	:	
125	7D		
126	7E		
127	7F	DEL	
251	FB	SB	Start buffer
252	FC	SVM	Secondary Value Mark
253	FD	VM	Value Mark
254	FE	AM	Attribute Mark
255	FF	SM	Segment Mark

SUMMARY OF THE PICK/BASIC DEBUGGER COMMANDS

The following is a summary of all the PICK/BASIC DEBUGGER commands and their descriptions.

Bx	Set breakpoint condition table where 'x' is a simple logical expression, which may be composed of < (less than), > (greater than), = (equal to), # (not equal to), & (and), and the special operator $ (line number).
D	Displays breakpoint and trace tables.
DEBUG	Escape to standard debugger.
DE	Short form of DEBUG.
En	Step on n+1 instructions. E [CR] turns mode off.
END	End execution of PICK/BASIC program and return to TCL.
G	Proceed from breakpoint.
Gn	Go to line n.
K	Kills all breakpoint conditions in table set by 'B' command.
Kx	Kills breakpoint condition 'x' where 'x' is the breakpoint number from 1-4.
Ln#	Display source code lines starting at n and continuing for # lines.
LP	All output forced to printer reverses status each time LP is selected.
Nx	Continue thru x+1 breakpoints before stopping.
OFF	Log off.
P	Inhibit PICK/BASIC program output.
PC	Printer close - output to spooler.
R	Pops return stack.
S	Display subroutine stack.
T	Turns breakpoint trace table alternatley off and on.
Tv	Set variable 'v' in trace breakpoint table.

U	Remove all breakpoint trace table variables set by 'T' command.
Uv	Remove breakpoint trace variable 'v' from table.
V	Verify object code.
Z	Request symbol table.
$	Current statement number.
?	Display current program name, line # and object code verification status.
/v	Print value of a variable 'v'.
/m(x)	Print value of a point 'x' in array 'm'.
/m(x,y)	Print value of point 'x,y' in array 'm'.
/m	Print the entire array where 'm' is the array.
/*	Dump entire symbol table.
[x,y]	String window where 'x' equals the start of the string and 'y' equals the length of the string. This command effects all outputs of variables and has no effect on input.
[	Removes string window (setting string length to zero has the same effect).
=	Equal sign prints out after the printing of a variable in any slash command except '/m'. The value of the variable may be changed at this point.

NOTE:

- Carriage return terminates all controls.
- A linefeed equals G [CR]
- Break key breaks to PICK/BASIC DEBUGGER from PICK/BASIC program at end of line.
- PICK/BASIC DEBUGGER prompts with '*'.

BASIC DEBUGGER MESSAGES

The following informative, warning or error messages are used by the BASIC DEBUGGER.

*E x	Single step breakpoint at line number 'x'.
*Bn x	Table breakpoint at line number 'x', 'n' equals number of breakpoint.
*V=x	Value of variable at breakpoint.
*Nvar	Variable not found in statement.
CMND?	Command not recognized.
NSTAT#	Statement number out of range of program.
SYM NOT FND	Symbol not found in table.
UNASSIGNED VAR	Variable not assigned a value.
STACK EMPTY	The subroutine return stack is empty.
STACK ILL	Illegal subroutine return stack format.
TBL FULL	Trace or break table full.
ILLGL SYM	Illegal symbol.
NOT IN TBL	Not in trace break table.
NO SYM TAB	Symbol table not in file.

SECTION 10

ICON/PICK SYSTEM MAINTENANCE

ICON®

Chapter 10

SYSTEM MAINTENANCE

THE PICK SYSTEM

USER MANUAL

10.1 VIRTUAL MEMORY STRUCTURE

PICK is a multi-programmable virtual memory machine with all of the virtual memory (i.e., disc) being directly addressable as if it were real memory (i.e., core).

The virtual memory of an PICK system resides on a magnetic disc drive, which is divided into 2048 byte "Frames". The frames are given Frame-ID's, or "FID"s numbered 1, 2, 3 ... up to the maximum FID, which depends upon the size of the disc.

The lower-numbered frames on the disc are "ABS" frames, which contain system software and workspaces. all frames above the ABS area are available for use in files. those frames not used in files make up the Available Space, sometimes called "Overflow".

EXECUTABLE AREA (ABS)

The ABS area consists of executable object code and process workspaces. Software written in PICK assembly language is loaded onto disc in the executable area. The length of the executable area is a system generation parameter, and must be between 513 and 1024. Frames 1 through 512 of the executable area are reserved for current and future PICK software. User-written assembly language programs may be place in frames found by using the FIND-FRAME verb, which identifies available frames.

WORK AREA

The PICK operating system allows multi-programming, which means more than one different program may be executed, on a time-sharing basis, by the CPU. each running program, or process, has a "Primary" workspace area of 32 contiguous frames, the first of which is called the "Process Control Block" (PCB).

The PCB of channel zero is normally frame 512 (200 hex). PCB's for succeeding processes are separated by 32, and therefore the PCB for channel one is 544 (220 hex), channel two is 240 hex, etc.

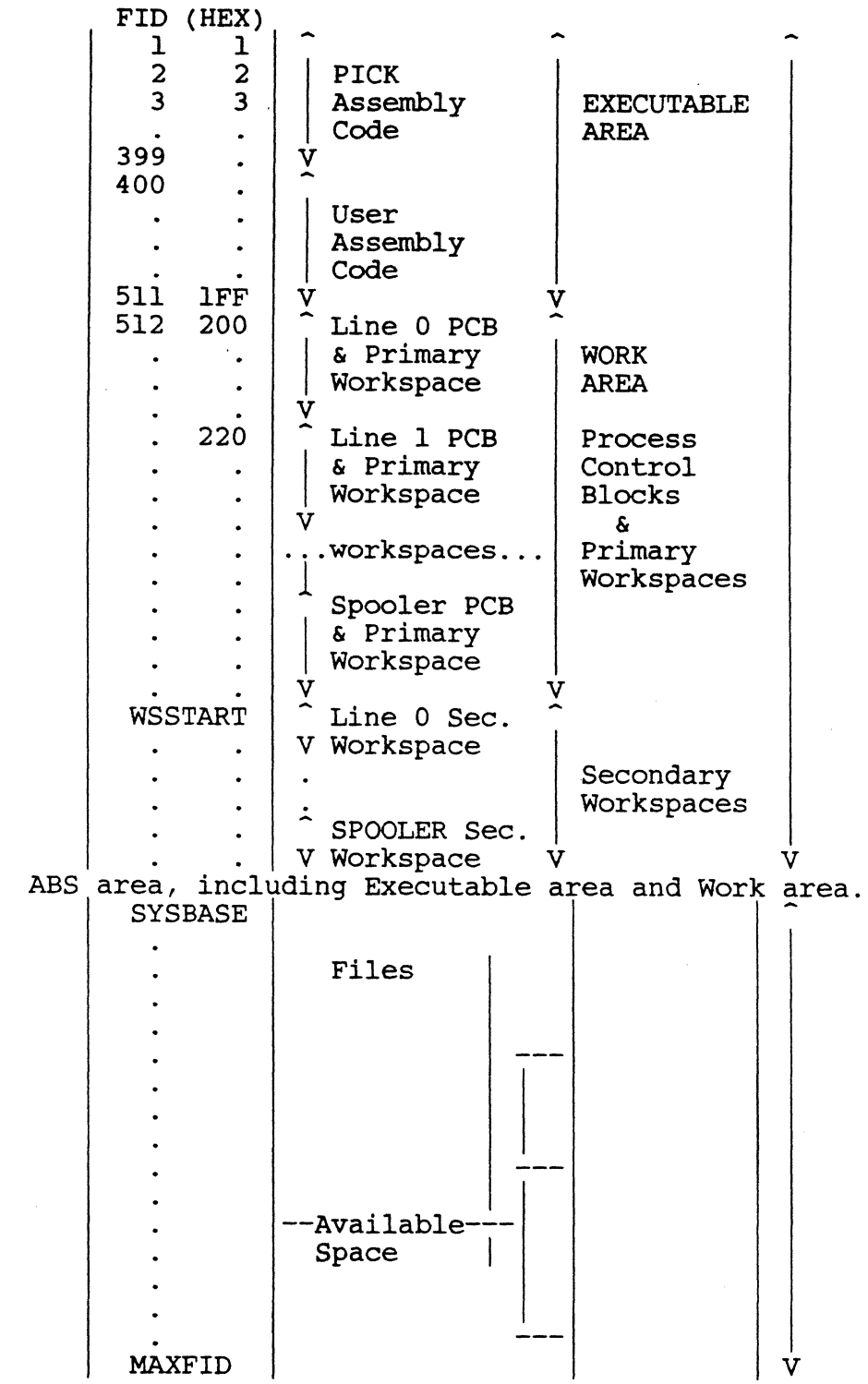
Additionally, larger "Secondary" workspace blocks are reserved following the last primary workspace, that of the SPOOLER. WSSTART is the starting FID of the secondary workspaces, which continue to the end of the work area. each process has three secondary workspaces, usually of 100 frames each.

FILES AND OVERFLOW

After the work area are the PICK files, beginning with the SYSTEM file. The base of the SYSTEM file, SYSBASE, is the beginning of the file space. on a newly generated or restored system, all other files on the system immediately follow the SYSTEM file. At the end of the files is the start of Available Space (overflow), which then continues until the end of the disc--MAXFID. (See the left side of the first figure.)

On a running system, the overflow area will become "fragmented", as frames

are taken from and returned to the overflow pool. (See the right side of the second figure.)



Files and Available Space, after a file-restore (left) and after undergoing normal fragmentation (right)

10.2 ADDITIONAL WORK-SPACE ALLOCATION

The "additional workspace" is a set of contiguous, linked frames that is initialized by the system at coldstart or system-generation time.

There are several processors in the system that require large amounts of workspace, or buffer area. This workspace is pre-assigned, and need not be linked up at LOGON time. The workspace is linked after a file-restore, or it may be linked from TCL by use of the LINK-WS verb. The SPOOLER process links the workspace for all the other lines, and no other user can log on the system while this linkage is taking place; the message:

LINKING WORK-SPACE; WAIT

will appear until the spooler has finished the linkage.

The starting FID of the workspace may be computed as below:

WSSTART = 512 + (number of lines)*32. Each line has three (3) workspaces of one hundred (100) contiguous frames.

The workspace may be linked on a live system using the LINK-WS verb on the SYSPROG account. This may be done if it is suspected that the links of the additional workspace have been destroyed for some reason. One manifestation of this situation is that BASIC programs may terminate with the "NOT ENOUGH WORK SPACE" message. Work-space links should be particularly suspect if a program or process aborts on one channel, but works correctly on others.

The general form of the verb to relink the workspace is:

LINK-WS {(n{-m})}

If the "(n)" or "(n-m)" is omitted, the workspace of ALL lines will be relinked, except those of lines logged on and that of the spooler process. The parenthetical specification may be used to limit the relinking process to lines "n", or "n" through "m" only.

As the linkage proceeds, the line-number of the process whose workspace is currently being linked is displayed on the terminal; if the line is logged on, the message "ON!" will be displayed, and THE WORK-SPACE IS NOT RELINKED!

The spooler's workspace can only be relinked via a coldstart!

Beginning immediately after the Work Area, the remainder of the virtual memory (called the File Area) is available for the storage of data in files. The portions of the File Area that are not allocated to the files are maintained as a pool of Available Space.

The beginning of the File Area is a system generation parameter. It may be computed via the following general formula:

```
Start of File Area = (FID of first PCB) +
((number of processes)*32 +
((number of processes)*(pre-assigned work-space)*3
```

Pre-assigned work-space is set to 100 frames per process per work-space. Each process (including the spooler) has 3 secondary workspaces of 100 frames each.

As an example, a system with 16 communication lines (therefore 17 processes including the spooler) will have the start of the file area at frame:

```
512 + (17 * 32) + (17 * 300) = 6156
```

The end of the File Area is the highest available disc frame, MAXFID.

File Area frames which are not allocated to the files are maintained as a pool of Available Space, often called "Overflow". Available Space is used by the Pick system file management routines as additional data space, as well as to other processors as scratch work space. The Pick Computer System maintains a table of pointers that define the Available Space, which may be either in a "linked" form, or in a "contiguous" form. Contiguous Available Space, as the name implies, consists of blocks of contiguous frames (defined by starting and ending numbers) that can be taken out of the pool either singly or as a block. Linked Available Space can only be taken a frame-at-a-time. Conversely, space may be released by processors to the linked available pool a frame-at-a-time, or to the contiguous pool as a block.

At the conclusion of a File-Restore process on the Pick system, an initial condition may be said to exist; there will be one principle block of contiguous available space, extending from the end of the current data space through the last available data frame. This is illustrated in the first figure; the results of the POVF (print overflow) verb indicate that there is no linked overflow space (blank line at the top of the output), and only one contiguous block of space.

As the system obtains and releases Available Space (and as files are created and deleted), the Available Space gets fragmented; at any particular time there may be several blocks of contiguous Available Space, and a chain of linked Available Space. Available frames will be placed in the linked Available Chain only when there are 31 sets of contiguous

Available space (representing the maximum that the system space management routines can maintain). This is illustrated in the second figure; here the linked Available chain starts at FID 23459 and contains 400 frames. There are also several sets of contiguous Available space as shown by the pairs of FIDs displayed.

Logically, there is no difference between Available space in linked chain and that in the contiguous sets; however, certain processors obtain frames from the contiguous space only, for example the CREATE-FILE processor, and the MEM-DIAG processor. Therefore, if the system Available space is severely fragmented, while there may be actually be enough disc space to create a large file, for example, there may not be enough available as a contiguous block. Periodically, a File-Restore may be run to restore contiguous Available space from the linked Available space chain.

(SEE: POVF)

```
>POVF [CR]

 23987- 97799

 TOTAL NUMBER OF CONTIGUOUS FRAMES AVAIABLE= 63812
```

Results of POVF immediately after a file-restore
(One contiguous block of Available space only)

```
>POVF [CR]
   23459 (400)
    8112-  8117 (    6)    9000-  9000 (    1)
   23789- 23801 (   13)   25000- 25678 (  679)
   25681- 25692 (   12)   27123- 27323 (  201)
   34502- 35123 (  522)   35800- 35801 (    2)
   37091- 37091 (    1)   37093- 37093 (    1)
   37099- 37100 (    2)   38100- 38100 (    1)
   43100- 44234 ( 1135)   45680- 45681 (    2)
   46343- 46443 (  101)   46445- 46445 (    1)
   46448- 46448 (    1)   46451- 46451 (    1)
   46454- 46454 (    1)   46458- 46474 (   17)
   47011- 47444 (  434)   47460- 47492 (   33)
   47661- 47750 (   90)   48012- 48017 (    6)
   48018- 48018 (    1)   48020- 48101 (   82)
   48233- 48268 (   36)   48299- 48299 (    1)
   51111- 53234 ( 2124)   53400- 53601 (  202)
   60000- 97799 (37800)
  TOTAL NUMBER OF CONTIGUOUS AVAILABLE FRAMES= 43509
```

Results of POVF after normal system operation.

A Frame is a block of 512 bytes that is referenced by a unique number called the Frame Identifier, or FID.

There are two types of frames in the Pick system - ABS frames, and FILE frames. The FILE frames contain only 500 bytes of data, the remaining 12 bytes being the "link fields". ABS frames may be object-code (assembly or PICK/BASIC-compiled object code), buffers or other workspaces required by the system.

Linked frames are used to define data areas which may be greater than 1 frame in length. The groups in data files may expand as more data is placed in the group, so when the end of a frame is reached, another frame is obtained from the system Available Space pool and linked to the end of the group. The format of the linked frame is as follows:

byte: 0	1	2	3	4	5	6	7	8	9	A	B	C
*	nncf	..forward link...				..backward link...				npcf	*	start of data

where:

* = Unused byte.

nncf = Number of next contiguous frames (count of frames that are linked forwards of this one, whose FID's are sequential to this FID).

npcf = Number of previous contiguous frames (count of frames that are linked backwards to this one, whose FID's are sequential to this FID).

forward link = FID of the frame that is next in logical sequence to this one.

backward link = FID of frame that is logically previous to this one.

The first frame of a linked set of frames will have zero "npcf" and "backward link" fields, and the last frame of such a set will have zero "nncf" and "forward link" fields. The "nncf" and "npcf" fields are also normally zero, except in the "linked workspace" allocated to each process, and in files that have a separation greater than one.

Following the link fields is the 500-byte data block.

Unlinked frames have no specified format; all 512 bytes of the frame may be used by the system.

10.5 DISPLAYING FRAME FORMATS; THE DUMP VERB

The DUMP verb may be used to display data in a frame. The data display may be specified in either character or hexadecimal format.

FORMAT:

DUMP nl{-n2},options

"nl" and "n2" are numbers that may be specified in decimal, or in hexadecimal by preceding the hex number with a period (.). nl and n2 contain the FID of the frame or the core location of the buffer being dumped. Options are specified just as normal statement options, single characters, optionally separated by commas. Valid options are specified in the table below:

OPTION	DESCRIPTION
G	Group; specifies that the data starting at frame nl is to be dumped, and that the dump continue following either the forward or backward links (depending on whether the U option is not or is specified). The dump will terminate when the last frame in the logical chain has been found.
L	Links; specifies that the dump be confined to the "links" of the frame(s) concerned; no data is displayed.
N	Nostop; if the data is printed on the terminal, specifies that the end-of-page stop be inhibited.
P	Printer; the display is routed to the line-printer.
U	The data or links are traced logically Upwards; that is, the backward links are used to continue the display.


```
>DUMP 6950,L [CR]

 FID:   6950 :   0  6967     0   0  (  1B26 :  0   1B37      0  0 )
+FID:   6967 :   0     0  6950   0  (  1B37 :  0      0   1B26  0 )
```

In the example above, the display indicates that 6950 is the FID whose links are being dumped; the "nncf" field is 0; the "forward link" field is 6967; the "backward link" field is 0; the "npcf" field is 0. Data in parentheses are the same numbers displayed in hexadecimal. The next line displays the link fields of FID 6967; the "+" indicates that this FID is logically "forward" of the preceding one.

```
>DUMP .1B37,X [CR]

FID:    6967 :   0     0  6950   0  (  1B37 :  0      0   1B26  0 )

0000  00000000 00000000 1B260000 2A2A2A2A      0 :.........&..****:
0010  2A2A2A2A 2A2A2A2A 2A2A2A2A 2A2A2A2A     16 :****************:
0020  2A2A2A2A 2A2A2A2A 2A2A2A2A 2A2A2A2A     32 :****************:
0030  2A2A2A2A 2A2A2A2A 2A2A2A2A 2A2A2A2A     48 :****************:
0040  2A2A2A2A 2A2A2A2A FE2A2020 54484953     64 :********^*  THIS:
0050  2050524F 4752414D 20464F52 4D415453     80 : PROGRAM FORMATS:
0060  20412044 4154412F 42415349 43205052     96 : A DATA/BASIC PR:
0070  4F475241 4D20544F 2020FE2A 20204449    112 :OGRAM TO  ^*  DI:
0080  53504C41 5920424C 4F434B20 53545255    128 :SPLAY BLOCK STRU:
0090  43545552 494E4720 42592049 4E44454E    144 :CTURING BY INDEN:
00A0  54494E47 204C494E 45532EFE 2A2A2A2A    160 :TING LINES.^****:
00B0  2A2A2A2A 2A2A2A2A 2A2A2A2A 2A2A2A2A    176 :****************:
00C0  2A2A2A2A 2A2A2A2A 2A2A2A2A 2A2A2A2A    192 :****************:
00D0  2A2A2A2A 2A2A2A2A 2A2A2A2A 2A2A2A2A    208 :****************:
00E0  2A2A2A2A 2A2A2A2A 2A2A2A2A FE2A2D2D    224 :************^*--:
00F0  2D2D2D2D 20444546 494E4954 494F4E53    240 :---- DEFINITIONS:
0100  FE313020 2020204C 4F4F50FE 20202020    256 :^10    LOOP^     :
0110  20202020 20535020 3D203620 20202020    272 :      SP = 6      :
0120  20202020 203B2A20 4C454654 204D4152    288 :      ;* LEFT MAR:
0130  47494E20 434F4C55 4D4E204E 554D4245    304 :GIN COLUMN NUMBE:
0140  52FE2020 20202020 20202049 44203D20    320 :R^           ID = :
0150  33202020 20202020 2020203B 2A204E55    336 :3            ;* NU:
0160  4D424552 204F4620 53504143 45532020    352 :MBER OF SPACES  :
0170  20544F20 494E4445 4E54FE20 20202020    368 : TO INDENT^     :
0180  20202020 464C463D 30202020 20202020    384 :    FLF=0       :
0190  20202020 3B2A2046 554E4B59 204C494E    400 :    ;* FUNKY LIN:
01A0  4520464C 4147FE2A 2D2D2D2D 20494E49    416 :E FLAG^*---- INI:
01B0  5449414C 495A4154 494F4EFE 20202020    432 :TIALIZATION^    :
01C0  20202020 20535058 203D2053 50FE2020    448 :      SPX = SP^  :
01D0  20202020 2020204C 494E452E 4E4F203D    464 :        LINE.NO =:
01E0  2030FE2A 2D2D2D2D 2D2D2049 4E505554    480 : 0^*------ INPUT:
01F0  2046494C 452D4E41 4D45203D 20224250    496 : FILE-NAME = "BP:
```

Sample usage of the DUMP verb.

10.6 THE SYSTEM FILE and SYSTEM-level FILES

The SYSTEM file is the highest-level file in the PICK file hierarchy. It contains the file-pointers to every account in the data-base, as well as pointers to the system-level files such as ACC, PROCLIB, etc.

Entries in the SYSTEM file define user M/DICT's or special files neccessary for the PICK software.

the M/DICT pointers are either D (file definition) or Q (file synonym) items. The item-ID's of such items are the USER-NAMES that the user enters when the system requests him to LOGON. Such items are created by the CREATE-ACCOUNT processor, (for D items,) or by use of the EDITOR or COPY processor for Q items. The format of user-identification items is discussed in the section on USER IDENTIFICATION ITEMS.

The SYSTEM file also contains the file-pointers to the system-level files that are necessary to the proper functioning of the system. These files are:

ACC	(Accounting file)
BLOCK-CONVERT	(for BLOCK-PRINT and PICK/BASIC (A) option)
POINTER-FILE	(Saved lists.)
PROCLIB	(Standard system PROC library)
SYSTEM-ERRORS	(Disc errors) SYSTEM-ERRORS

The ACC file (Accounting history) has two types of items; those that indicate the actively logged-on users, and the accounting-history data items that keep track of the usage statistics of each user. The format of the items in this file is discussed in later sections.

The ACC files have a tri-level structure, with an ACC account, an ACC dictionary and an ACC data section.

The BLOCK-CONVERT file contains two unrelated types of items:

1) Items that define the format used in the characters displayed when the BLOCK-PRINT verb is used.

2) Items that are used to print a descriptive message when the "A" (assembly-code) option is used when compiling a PICK/BASIC program.

The BLOCK-CONVERT file is a single-level file.

The POINTER-FILE contains items that are "pointers" to binary data strings. It is referenced implicitly whenever the SAVE-LIST, GET-LIST, DELETE-LIST, CATALOG or DECATALOG verbs are used. The POINTER-FILE is a single-level file. The file-defining entry "POINTER-FILE" in the SYSTEM file must have the code "DC" in line 1. This indicates that the file contains non-standard, binary data items.

The PROCLIB file is a single-level file that contains commonly used PROCs, such as CT (Copy to Terminal), LISTU (List active users), etc.

The SYSTEM-ERRORS file is a three-level file reserved for logging system errors. currently its only use is to store disc errors.

Level 0	SYSTEM dictionary				
Level 1	ACC (account)	BLOCK-CONVERT dictionary	POINTER-FILE dictionary	PROCLIB dictionary	SYSTEM-ERRORS (account)
Level 2	ACC dictionary				SYSTEM-ERRORS dictionary
Level 3	ACC data				SYSTEM-ERRORS data

SYSTEM-level files

10.7 THE BLOCK-CONVERT AND POINTER-FILE DICTIONARIES

This section describes the format of entries in the BLOCK-CONVERT, and POINTER-FILE dictionaries.

BLOCK-CONVERT dictionary:

There are two types of entries in the BLOCK-CONVERT file; type I is the entry that forms the characters for the BLOCK-PRINT verb. Its format is:

Item-id: is the character to be formed; that is, the item whose item-id is "C" will form the character C; that whose item-id is "{" will form the character {, etc.

The first attribute contains a code of the form:

n{c}

where "n" is the width of the character matrix (the depth of all characters formed is fixed at 8); and the optional "c" is a character that will replace the item-id in the generation of the BLOCK-PRINT form.

There must be 8 succeeding attributes, each one specifying the format of a row in the generated form. Each attribute must begin with a "C" or a "B", specifying a character insertion or a blank insertion respectively, followed by the number of such insertions needed; optionally, additional numbers may be specified, separated by commas. Each succeeding number switches the insertion from character to blank and vice-versa. The sum of all numbers must equal the character width specified in attribute 1. See the first example.

The second type of data in the BLOCK-CONVERT file has a two-hexadecimal-digit item-id, corresponding to the PICK/BASIC opcode generated by the PICK/BASIC compiler; attribute 1 is the symbolic name for the opcode. These entries are used by the "A" option of the PICK/BASIC compiler to generate a listing of the PICK/BASIC object code.

POINTER-FILE dictionary:

This file contains the pointers to select-lists (stored by the SAVE-LIST verb) and to cataloged PICK/BASIC programs (stored by the CATALOG verb). They may be examined, but, like file-pointers, should never be altered in any way by the user!. The format of these items is:

Item-id	account-name*x*y	where "x" is C for a cataloged program, or "L" for a select-list, and y is the program-name, or select-list name.
001	CL or CC	CL for lists, CC for programs.
002	fid	Base FID of the program or list.
003	n	# frames in the program or list.
004	m	Number of items in a list; null for a program.
005	time & date	Time and date of generation.


```
>COPY BLOCK-CONVERT S 8B (T) [CR]

    S    Item-id; defines format for character "S"
001 7    Defines character width as 7
002 B1,5,1                          Specifies string " SSSSS " (1 blank, 5 S
003 C2,3,2                          Specifies string "SS   SS"
004 C2,5
005 B1,5,1                          Specifies string " SSSSS "
006 B5,2
007 C2,3,2
008 B1,5,1
009 B7

    8A   Item-id (BASIC object-code byte)
001 STOP Identifies object-code (STOP opcode).
```

Sample items from BLOCK-CONVERT file.

```
>BLOCK-PRINT S [CR]

 SSSSS
SS   SS
SS
 SSSSS
     SS
SS   SS
 SSSSS
```

Output using BLOCK-PRINT verb.

Most error messages generated by TCL, ACCESS, PICK/BASIC, PROC or any other system software are contained in the ERRMSG file. a standard set of approximately 250 error message items is provided with the PICK base system. however, the user may change the error messages in the ERRMSG file, add new error messages, or even create another ERRMSG file for each account. this can be particularly useful when used in conjunction with the STOP and ABORT statements in PICK/BASIC, in which the user can specify an error message and pass parameters to the error message processor.

Items in the ERRMSG file must follow a certain format, in which the first character in each line of the item defines a special operation, as listed below.

CHARACTER	MEANING
H	Causes the string following the "H" on be placed in the output buffer, with no carriage return or line feed. At the end of the error message item, the string "H+" will inhibit the final carriage-return/line-feed that is normally output.
L	Causes the output buffer to be printed, with a carriage-return and line-feed
L(n)	As above, and also causes n-1 blank lines to be printed.
D	Places the current date in the output buffer.
T	Places the current time in the output buffer.
A	Inserts the next parameter in the list of parameters which was passed to the error message processor with the error message. The parameters may be specified by the PICK/BASIC program (in the case of a PICK/BASIC STOP or ABORT statement,) or by some system processor in the case of system-generated error messages.
R(n)	Inserts the next parameter right-justified in a field of n blanks.
A(n)	Same as R(n), but left-justified.
X	Skips a parameter in the parameter list.
S(n)	Sets the output buffer pointer to location "n".

SPECIAL ERRMSG FILE ITEMS. The item "LOGON" in the SYSTEM dictionary contains the request to logon to the system (typically "LOGON PLEASE").

When a user logs onto an PICK system, the error message specified by the item "LOGON" in the ERRMSG file is printed on the user's terminal. Therefore, any message which is to be received by all users on the system immediately upon logging on may be placed in this item. This item must exist on file even if there is to be no general system message.

The ERRMSG items "335" and "336" contain the connect time messages displayed when a user logs on or off the system.

Some examples of error message processing are shown in the first figure. The PRINT-ERR verb allows the user to invoke the error message processor from TCL. the format is:

>PRINT-ERR file-name item-list

the error messages specified in the item-list will be processed, with a parameter list of A,B,C,D... See second figure.

In a PICK/BASIC program, the lines...

```
FILE = "BP" ; ID = "1006"
OPEN "",FILE ELSE STOP 201,FILE
READ ITEM FROM ID ELSE STOP 202,ID
```

could cause the program to stop with either of the following:

```
[201] 'BP' IS NOT A FILE NAME
'1006' NOT ON FILE.
```

If the item "LOGON" in the ERRMSG file for an account looked like:

```
HHello out there!
L
HIt's now
T
H and all's well!
```

then the user would see the following when he logged on:

```
Hello out there!
It's now 11:22:33 and all's well!
```

Sample Usage of the Error Message Processor.

```
>PRINT-ERR ERRMSG 201 [CR]
[201] 'A' IS NOT A FILE NAME

>PRINT-ERR ERRMSG 289

             TERMINAL PRINTER
PAGE WIDTH:      A        B
PAGE DEPTH:      C        D
LINE SKIP :      E
LF DELAY  :      F
FF DELAY  :      G
BACKSPACE :      H
TERM TYPE :      I
```

Sample Usage of the PRINT-ERR verb.

10.9 USER IDENTIFICATION ITEMS

Each user has a user identification item stored in the System Dictionary (SYSTEM). This set of user identification items define the users who can log on to the system. User identification items are either file definition items or file synonym definition items.

User identification items are initially created via the CREATE-ACCOUNT PROC These items may subsequently be updated via the EDITOR. ENTRIES IN THE SYSTEM DICTIONARY SHOULD NOT BE UPDATED (from the SYSPROG account) WHEN ANY OTHER USER IS LOGGED ON to the system. this is because the system software maintains pointers to data in the System Dictionary when users log on, and updating the System Dictionary will invalidate the pointers. An exception to this rule is when creating a new account (or a synonym to an existing account), which can be done at any time since new items are added to the end of the existing System Dictionary data, and thus do not disturb any pointers to it.

Attributes five through eight of a user identification item contain data associated with the user's security (lock) codes, password, and system privileges:

ATTRIBUTE	USE
5	Contains the set of retrieval lock-codes associated with the user. Multiple values (separated by value marks) are allowable. There is no restriction as to the format of individual lock-codes. This attribute may be null, indicating no lock-codes. (Lock-code usage is described in the topic titled SECURITY.)
6	Contains the set of update lock-codes associated with the user. (Same as described for retrieval lock-codes above.)
7	Contains the user's password, which is a single value. This attribute may be null. There is no restriction as to the format of the password.
8	Contains a code which indicates the level of "system privileges" (see below) assigned to the user.
9	May contain the code "U" to indicate that logon/ logoff times should be logged by the system. May contain the code "R" to specify the RESTART option.

Attributes one through four and attributes ten through thirteen are as defined for regular file definition of file synonym definition items (see topic titled DICTIONARIES). The first figure shows a sample user identification item (for user SMITH).

Three levels of system privileges are available; they are referred to as zero (lowest), one, and two (highest), respectively. Lower levels of system privileges restrict usage of certain facilities of the system, as described in the second figure. System privileges are assigned by the code in attribute eight of the user identification item. Leave this part null for level 0, SYS1 for level 1, and SYS2 for level 2.

Attribute 9 may contain the codes 'U' or 'R', or both. 'U' specifies that the accounting listing file is to be updated whenever the user logs on and off this account (see ACCOUNTING FILE). 'R' specifies that the Restart option is to be set. This causes the LOGON PROC to be re-executed whenever an "END" is typed at the DEBUG level.

```
Item SMITH in System Dictionary

001 D <------------------------- D/CODE
002 2537 <---------------------- Base FID
003 13 <------------------------ Modulo
004 1 <------------------------- Separation
005 ABC <----------------------- Retrieval lock Code (L/RET)
006 1234 <---------------------- Update Lock Code (L/UPD)
007 PW5 <----------------------- Password
008 SYS2 <---------------------- System Privilege Level
009 U <------------------------- Update Account File for this user
```

Sample User Identification Item (For User SMITH)

FACILITY	LOWEST PRIVILEGE LEVEL REQUIRED
Updating of M/DICT	One
Use of magnetic tape	One
Use of DEBUG (other than P, OFF, END and G commands).	Two
Use of DUMP Processor	Two
Use of Assembler and Loader	Two
Use of FILE-SAVE and FILE-RESTORE processors.	Two

Required System Privilege Levels.

Security codes may optionally be placed in the L/RET and L/UPD attributes of a dictionary item to restrict access and update. At Logon time, each user is assigned the set of security codes which are in his user identification item. During the session, whenever, an L/RET or L/UPD code is encountered, a search is made of the user assigned codes for a match; if no match is found the user is denied access. A security code may consist of any combination of legal ASCII characters.

L/RET and L/UPD

Both file definition ("D" code) and synonym file definition ("Q" code) items have L/RET (retrieval locks) and L/UPD (update locks) attributes. When these attributes have values stored, they are known as security codes. Although there is no prohibition against multiple values for these attributes, only the first attribute value is used for matching against the user assigned codes. Since each file may be individually locked for both update and retrieval, a user must be assigned multiple codes to that set of data he is allowed to access. Using this feature, a complex "mask" can be constructed for each user, giving each user a different sub-set of files which he may access.

Security at the file level is invoked at the processor level. The following processors are assumed to be updating processors and therefore require a match on the L/UPD attribute in the file definition item: COPY, EDIT, PICK/BASIC if writing a file, RUN and the Assembler. Other processors are assumed to be retrieval processors and require a match on the L/RET attribute in the file definition item.

PICK/BASIC requires a match against L/RET code when the file is opened; and requires a match against the L/UPD if data is changed in the file.

Failure to match one of the user security codes with either the L/RET (or L/UPD) attribute value will generate the following message (and return control to TCL):

```
[210] FILE xxx IS ACCESS PROTECTED
```

User Assigned Codes

Each user identification item in the System Dictionary (see topic titled USER IDENTIFICATION ITEMS) contains the list of security codes assigned for that particular user. These codes are values for the attributes L/RET and L/UPD. The lock code in the user-identification item and the lock code in the file being verified must match.

Security codes may be assigned initially when an account is created via use of the CREATE-ACCOUNT PROC Security codes may be added or deleted by updating the appropriate security codes); however, updates to the user identification item should only be performed when no one else is logged onto the system.

Security Code Comparison

Security codes are verified comparing the value in the file dictionary against the corresponding string of values in the user identification item. Characters are compared from left to right. An equal (verified) compare occurs when the value in the file dictionary is exhausted and all characters match up to that point. This is illustrated below.

When referencing a file using a Q synonym a security code match is made at all levels (i.e., SYSTEM, M/DICT, and file dictionary) and therefore a correspondence must be maintained at all levels in order to process the Q synonym files. Since the user identification item for the account containing the primary file is verified for security codes, the user referencing the Q synonym must have a code defined in this user identification item which will verify with the first code in the equated account's user identification item. Therefore, in a user identification item, only the first code is used to protect the account from Q synonym accesses,while all the codes in the item are assigned to the user when he logs on.

FILE DICTIONARY CODE	USER IDENTIFICATION CODE	RESULT
123	123	Match
12	123	Match
123	12	No Match
XYZ	XYZ5	Match
AQ2	AQ	No Match

Sample Security Code Comparisons.

10.11 THE ACCOUNTING HISTORY FILE: AN INTRODUCTION

The Accounting History File is one of the mandatory files in the PICK system. This file contains accounting history for the system, as well as the entries that describe the currently active (logged-on) users.

The System dictionary (SYSTEM) contains the file definition item (item-id 'ACC') for the Accounting History File, as illustrated in the figure. The 'ACC' dictionary is set up for examining and listing the data in Accounting History File via ACCESS (see topic titled THE ACCOUNTING HISTORY FILE: SUMMARY AND EXAMPLES). There are two types of entries (items) in the Accounting History File: those that represent active (logged-on) users, and those that keep track of accounting history.

Active Users Items

The item-id of an active user item in the Accounting History File is the four-character hexadecimal FID of the PCB of the user's process. If the PCB's start at FID=512, (they proceed in steps of 32 frames from there on), we see that a user logged on to process zero will have an entry with an item-id '0200' (512), while a user logged on to process one will have an entry with an item-id '0220' (544), and so on. Attribute one of an active user item contains the name of the user (i.e., the item-id of the user identification item), attribute three the date logged on, and attribute four the time logged on. Active user items are created when a user logs on, and deleted when he logs off.

Accounting History Items

The item-id of an accounting history item is the name of the user (i.e., the item-id of the user identification item), with the channel number concatenated by a "#". For example, if user 'SMITH' logs on to channel 12, when he logs off, the item whose item-id is 'SMITH#12' in the ACC file will be updated. This allows one to keep track of system usage by user-id as well as channel number.

Attributes one, two and three are not used. The remainder of the attributes are described below:

ATTRIBUTE	USE
4	Dates(s) Logged on. Each unique date is stored. Value marks are tagged on to the value in this attribute if multiple Logoffs occur on the same date (for LIST alignment purposes). Date is stored in Pick Computer System date format.
5	Time(s) Logged on. An entry is made for each Log-off, representing the time at which the user Logged on. Time is represented in seconds past midnight (24- hour clock).
6	Connect time(s). This entry represents the time in seconds between the Logon and the Logoff.
7	Charge-units. A number representing the CPU usage is added on each Logoff.

8 Line-printer pages. A number representing the number of pages routed to the line-printer for each session.

Note: Attributes 4, 5, 6, 7 and 8 are stored as a "Controlling-dependent" data set, with attribute 4 being the controlling value, and the others the dependent ones. See the ACCESS reference manual for a discussion of the "controlling-dependent" data set format.

The accounting history file 'ACC' is not automatically updated every time a user logs off the system. The SYSTEM dictionary item for the user must have a 'U' in attribute 9 if the user is to have his Account file history items updated. The entries in the Account file contain the history of each session (logon to logoff). If the SYSTEM dictionary data has been changed since logon or the history item to the updated is too large for the work-space, the message number 338 will be printed.

Channel #	Item-id	Channel #	Item-id
0	0200	16	0400
1	0220	17	0420
2	0240	18	0440
3	0260	19	0460
4	0280	20	0480
5	02A0	21	04A0
6	02C0	22	04C0
7	02E0	23	04E0
8	0300	24	0500
9	0320	25	0520
10	0340	26	0540
11	0360	27	0560
12	0380	28	0580
13	03A0	29	05A0
14	03C0	30	05C0
15	03E0		

Channel (Line) numbers and corrosponding Active User Item-IDs.

10.12 THE ACCOUNTING HISTORY FILE: SUMMARY AND EXAMPLES

This topic summarizes the formats of the active user items and the accounting history items in the Accounting History File. Also presented are sample entries for the Accounting History File.

The first figure summarizes the attributes for the active user items and the accounting history items. The second figure shows a sample sorted listing of the active users (users with a value for attribute A1) via an ACCESS SORT statement. The third figure shows a sample listing of the accounting history item for user SMITH via an ACCESS LIST statement.

ATTRIBUTE NUMBER	'ACC' DICTIONARY NAME	ACTIVE USER ITEM	ACCOUNTING HISTORY ITEM
	(item-id)	Four-character hexadecimal PCB-FID	User name#lineno
1	NAME	User name	Not used
2	DATE	Date logged on	Not used
3	TIME	Time logged on	Not used
4	DATES		Dates logged on
5	TIMES		Times logged on
6	CONN		Connect time
7	UNITS		Charge-units
8	PAGES		Number of printer pages generated.

Summary of Active User Items and Accounting History Items


```
>LISTU [CR]

CH# PCBF NAME..... TIME... DATE.... LOCATION.....

 00 0200 CM              11:02AM 03/22/78 Channel 0
 01 0220 SYSPROG         12:10PM 03/22/78 Channel 1
 02 0240 EL-ROD          09:11AM 03/22/78 Channel 2
 03 0260 LC              06:59AM 03/22/78 Channel 3
 05 02A0 HVE             09:55AM 03/22/78 Channel 5
*06 02C0 CM              11:25AM 03/22/78 Channel 6
 07 02E0 BUGEYE          01:29PM 03/21/78 Channel 7
 10 0340 JT              11:34AM 03/22/78 Channel 10
```

Sample sorted listing of active user items (using LISTU).

```
>LIST ACC = "SMITH]"                (selects items with item-ids
                                      starting with the string "SMITH")
PAGE   1                                        12:17:22  22 MAR 1978

ACC..... DATE. TIME... CONN... UNITS.. PAGES
               *       *       *       *
SMITH#0  01/13   16:56   00:04       9
         01/14   10:13   00:00       5
                 10:15   00:01     343
         02/06   17:02   00:18      41
         02/09   10:21   00:17     690
         02/23   07:58   00:01      27
         03/09   11:35   01:57     378
                 16:05   00:22      94
SMITH#5  01/13   12:48   02:25     160      5
                 15:20   00:05      14
                 15:25   00:00       2
                 15:28   00:17     110
                 16:20   02:55    2575     16
                 19:15   00:00      13
         01/16   09:41   06:13    1853      6
                 15:55   00:12      15

2 ITEMS LISTED.
```

Sample listing of accounting-history item for user "SMITH".

To avoid overflowing the accounting history items in the Accounting History file for a specific user, the items should be periodically cleared.

To clear the accounting history items from the ACC file, follow the steps detailed in the first figure.

The point of overflow is determined by the activity of the user-account (however, approximately 1000 Logon/Logoffs are allowed). This point can be calculated by following the procedure detailed in the second figure.

If the accounting history item for a user-account does exceed the available workspace, the user will be logged off, but the Accounting History File will not be updated. To recover from this situation, follow the procedure detailed below.

1. Logon to the SYSPROG account.
2. Type the following (if you need a listing only):

```
>SORT ACC WITH NAME LPTR [CR]
```

3. Type the following:

```
>SELECT ACC WITH NAME [CR]
>DELETE ACC  [CR]
```

Procedure to Clear all Accounting History Items.

1. Use the WHAT verb to determine the number of additional work-space frames allocated for the system (parameter WSSIZE in the WHAT display). Multiply this figure by 500 and add 3000.
2. To determine the current size, type:

```
>STAT ACC ACC-SIZE 'user-name'  [CR]
```

This will produce the following output:

```
STATISTICS OF ACC-SIZE:
TOTAL = xxx AVERAGE = yyy COUNT = zzz
```

3. If the value displayed for TOTAL in step 2 (i.e., xxx) approaches the value calculated in step 1, then the user-account is approaching the overflow point.

Determining the point of overflow for an accounting-history item.

The ITEM and GROUP commands provide information about the item and group structure of Pick files. Output can be displayed at the terminal or optionally directed to the line printer.

FORMAT:

ITEM file-name item-id {(options)}

This command displays the base FID of the group into which the specified item-id hashes. If the item is not already on file, the message "ITEM NOT FOUND" is displayed. In addition, every item-id in that group is listed along with a character count of the item (in hex). At the end of the list the following message is displayed:

n ITEMS m BYTES p/q FRAMES

where:

n is the number of items in the group.
m is the total number of bytes used in the group.
p is the number of full frames in the group.
q is the number of bytes used in the last frame of the group.

Valid options for this command are as follows:

P - Direct output to line printer.
S - Suppress item list.

FORMAT:

GROUP file-name {(options)}

This command displays the base FID of each group in the specified file. In addition, every item-id in the group is listed along with a character count of the item (in hex). At the end of the list for each group the following message is displayed:

n ITEMS m BYTES p/q FRAMES

where:

n is the number of items in the group.
m is the total number bytes used in the group.
p is the number of full frames in the group.
q is the number of bytes used in the last frame of the group.

Valid options for this command are as follows:

P - Direct output to line printer.
S - Suppress item list.

```
>ITEM M/DICT A [CR]


27121
0022 FILE-DOC
001C bd
0009 A
0011 T-ATT
000F DUMP
0018 B/ADD
000F DIVX
0014 EDIT-LIST
0028 V/CONV
0022 LISTU
0019 V/MIN
001B ACCOUNT-RESTORE
001D D/CODE
0028 SL
0023 INST-INDEX
0047 SAL
0072 TB
000E SAVE
18 ITEMS 591 BYTES 1/91 FRAMES
```

Figure A. Displaying data in a group using the ITEM command.

ISTAT and HASH-TEST are ACCESS verbs that produce file hashing histograms, ISTAT for specified file items and HASH-TEST on the basis of a user-specified test modulo.

ISTAT

An ACCESS sentence using the ISTAT command is constructed as illustrated below. The ISTAT command provides a file hashing histogram for the selected items in the selected file, as illustrated by the examples. For further information regarding file hashing, refer to the section of this manual titled VIRTUAL MEMORY OPERATING SYSTEM.

HASH-TEST

HASH-TEST produces a file hashing histogram as a result of a user-specified test modulo. The general form of this verb is as follows:

HASH-TEST {DICT} file-name {item-list} {selection-criteria}

```
>ISTAT PARCEL [CR]

FILE= PARCEL MODULO= 3 SEPAR= 1                13:50:42  22 MAR 1978
FRAMES BYTES ITMS
000002 00757 027 *>>>>>>>>>>>>>>>>>>>>>>>>>>>
000002 00836 030 *>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
000002 00785 028 *>>>>>>>>>>>>>>>>>>>>>>>>>>>

ITEM COUNT=          85, BYTE COUNT=      2378, AVG. BYTES/ITEM=     27.9
AVG. ITEMS/GROUP=  28.3, STD. DEVIATION=   1.5, AVG. BYTES/GROUP=   792.6.
```

Sample usage of the ISTAT command.

```
>HASH-TEST PARCEL [CR]

TEST MODULO: 9 [CR]
FILE= PARCEL MODULO= 9 SEPAR= 1                13:50:55  22 MAR 1978
FRAMES BYTES ITMS
000001 00256 009 *>>>>>>>>>
000001 00281 010 *>>>>>>>>>>
000001 00255 009 *>>>>>>>>>
000001 00229 008 *>>>>>>>>
000001 00248 009 *>>>>>>>>>
000001 00251 009 *>>>>>>>>>
000001 00272 010 *>>>>>>>>>>
000001 00307 011 *>>>>>>>>>>>
000001 00279 010 *>>>>>>>>>>

ITEM COUNT=          85, BYTE COUNT=      2378, AVG. BYTES/ITEM=     27.9
AVG. ITEMS/GROUP=   9.4, STD. DEVIATION=    .8, AVG. BYTES/GROUP=   264.2.
```

Sample usage of the HASH-TEST verb.

10.16 DETERMINING NATURE OF GROUP FORMAT ERRORS

10.16.1 GROUP DEFINITION

The term group is used to specify one 'bucket' of storage. A file is made up of a collection of groups, such that there are the same number of groups as the number specified for the modulo of the file. Put another way, the modulo of the file specifies the number of groups which make up the file.

The hashing algorithm takes the specified item-id and decides in which group it is or should be stored. The file retrieval or storage routine then searches that group for the specified item. The hashing algorithm may be thought of as dividing the item-id by the modulo in order to obtain the remainder. This remainder is then the 'group number', and specifies the group which is to be searched.

Within each group the items are stored physically end to end. Each item is made up of a count field, a key, and the data. The documentation for this system has conventionally used the term 'item-id' in place of the term 'key'. It remains that the item-id is the key which is used to look up the location of the item.

The count field exists only in a file representation of the item. It is a sixteen-bit binary number, such that the high-order bit is zero, represented in the file in ASCII hexadecimal notation, and as such takes up four bytes of storage. It immediately precedes the item-id in the file. If the item in question is the first item in the group, the count field starts in the first data byte in the frame. If the item is not the first item in the group, then the count field starts at the first byte after the termination mark of the last item.

The count field is used as a pointer to the end of the item. The end of the item must be an attribute mark followed by a segment mark. If the count field does not point to this pattern, there is a group format error, and the group format error handler will be entered.

10.16.2 GROUP FORMAT ERRORS

A GROUP FORMAT ERROR IS THE RESULT OF A HARDWARE ERROR!

A group format error is sensed if the count field does not point at an attribute mark, segment mark sequence. This may occur if the count is wrong, or if the data at the end of the item is wrong.

The count field is definitely wrong if any of the four digits which make up the count field are not ASCII hexadecimal digits, which are X'30' - X'39' or X'41' - X'46', which are 0-9 and A-F.

The end of item data may be wrong if the count field contains the wrong ASCII hexadecimal digits, or if the end of item data is actually wrong.

The end of item data may be wrong in several ways. If the item is contained in a frame, then the end of item data may be wrong in the ways that the the count field may be wrong. If the item spans a frame boundary, certain other mechanisms come into play. If a process was in the process of updating an item, to the extent that the first frame containing the item was written to disc, but that the last frame was not written when the process was interrupted by something like a cold start, then a group format error will occur. If the overflow handler becomes confused, the frames attached to a group may be acquired by another data file or by a print file. The difference should be obvious on inspection, using the DUMP verb. Print files do not normally contain attribute or value marks and data files do not normally contain carraige-return, line-feed sequences.

If the damaged frame is the result of an incomplete update, then the difficulty is localized. Repair of this group will usually attend to the matter. If the damage appears to be due to co-ownership of the frame, the problem may be greater. In this case it is best to leave the frame with the frame to which it has a back-link, presuming that the data is consistent in that chain. Then cut the forward link in the spurious chain and terminate the group.

The effect of the group format error handler is to terminate the group at the end of the last consistent item and cut the forward link out of the last acceptable frame in the group. The rest of the overflow is intentionally lost, because of the effect of having two copies of the same frame referenced in the overflow chain.

The one case in which the group will not be terminated is when a print file has meandered across the base of the file. In this case it is probably best to recreate the file and selectively restore it. The old file pointer should be thrown away. Do not use the DELETE-FILE verb on the old file, because this will further muddy the condition of the overflow handler.

10.16.3 RECOVERY FROM GFE's

If a group format error is encountered, the system will invoke the group format error handler. This processor will print the error message to the terminal and wait for an operator response. The valid operator responses are:

'D' - which will enter the system debugger.

'E' - which will end the process and return to TCL.

'F' - which will allow the GFE handler to fix the error and continue.

NOTE that fixing the error will undoubtedly cause the loss of at least one data item. This record normally must be manually recovered! The recovery strategy is to identify the file affected and do a SEL- RESTORE on the file. It is best to do this as soon after the group format error is noticed as possible.

10.17 GENERATING CHECKSUMS: THE CHECK-SUM COMMAND

The CHECK-SUM command generates a checksum for file items, thus providing a means to determine if data in a file has been changed.

FORMAT:

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

A checksum is generated for items in the specified file, or subset of items if the optional "item-list" and/or "selection-criteria" appear. Furthermore, the checksum may be calculated for one specified attribute. If no attribute is specified, the 1st default attribute will be used. If there is no default attribute, or if the AMC is 9999, the entire item will be included. The checksum will include the binary value of each character times a positional value. This yields a checksum which has a high probability of being unique for a given character string. The dictionary portion is checksummed if the "DICT" option appears. (A checksum is the arithmetic total, disregarding overflow, of all bytes in the selected items.)

A message is output, giving checksum statistics, in the following form:

```
BYTE STATISTICS FOR  file-name (or attribute name):
TOTAL = t AVERAGE = a ITEMS = i CKSUM= c BITS = b
```

where:

t is the total number of bytes in the attribute (or item) included
a is the average number of bytes
i is the number of items
c is the checksum
b is a bit count

The attribute mark trailing the specified attribute (or item) will be included in the statistics.

To use checksums, the user should issue CHECK-SUM commands for all files, or portions of files, to be verified and keep the output statistics. Subsequently, the CHECK-SUM commands can be reissued to verify that the checksum statistics have not changed. The checksum must be recalculated whenever the user updates the file!

10.18 SYSTEM PROGRAMMER (SYSPROG) ACCOUNT

Several special faciities are normally used from the System Programmer (SYSPROG) Account. Procedures on this account are normally performed by persons more familiar with the overall operation of the system.

To log on to the SYSPROG Account, type the following:

LOGON PLEASE: SYSPROG,password [CR]

where "password" is the appropriate password set up for SYSPROG. Alternate logon names (such as SP) may be used.

CREATE-ACCOUNT	DELETE-ACCOUNT
ACCOUNT-RESTORE	SAVE
BUFFERS	SEL-RESTORE
LOCK-FRAME	UNLOCK-FRAME
:FILES	:ABSLOAD
:ABS/FILES	WHAT

Some SYSPROG Verbs and Procs.

10.19 AVAILABLE SYSTEM SPACE: THE POVF COMMAND

The POVF verb displays the system overflow table.

FORMAT:

POVF {(P}

The POVF verb displays the contents of the system overflow table.

The P option forces all printed output to the line printer. the first line of output is the FID of the first frame in linked overflow, folowed by the number of frames in the linked chain. the next lines (up to 16) describe blocks of contiguous overflow, and have the following format:

m - n : p m - n : p

where:

m is the first frame of a contiguous block.
n is the last frame of the block.
p is the number of frames in the block.

The total number of frames contained in all the contiguous overflow is then printed (using error message number 293):

TOTAL NUMBER OF CONTIGUOUS FRAMES :number

The CREATE-ACCOUNT PROC is used to create new user-accounts. The SETUP-ASSY PROC enables the user account to assemble Pick programs.

CREATE-ACCOUNT PROC

The CREATE-ACCOUNT PROC generates a new account according to given specifications. It then copies the contents of the NEWAC file (the prototype M/DICT to the new user M/DICT. Finally it adds a file synonym (Q item) to the account into SYSPROG's M/DICT.

The CREATE-ACCOUNT PROC is invoked by typing in the PROC name:

```
>CREATE-ACCOUNT  [CR]
```

The PROC then prompts the user for the required information, as shown below.

The CREATE-ACCOUNT PROC should not be used to create a new synonym to an existent account; this should be done by using the EDITOR to create the file synonym definition item (Q-item) in the SYSTEM dictionary.

SETUP-ASSY PROC

The SETUP-ASSY PROC sets up a user-account so that it is able to assemble Assembly Language programs. To invoke this PROC enter "SETUP-ASSY" followed by the account name. For example:

```
>SETUP-ASSY USER3  [CR]
```

Once invoked, the PROC will ask a series of questions requiring input from the user. These questions are self-explanatory.

```
 >CREATE-ACCOUNT          PROC is typed in at TCL.
 ACCOUNT NAME?SHERRY      Anything but [CR] is legal.
 L/RET CODE(S)?AAA]BBB    Multi-valued retrieval code.
L/UPD CODE(S)?           [CR] means no lock code.
 PASSWORD?R2D2            User's LOGON password.
PRIVILEGES?              [CR] means SYS0. May be SYS0, SYS1, or SYS2.
 MOD, SEP?37,1            [CR] defaults to 29,1.

FILE 'SHERRY'           CREATED; BASE= 34593 MODULO= 37 SEPAR = 1

391 ITEMS COPIED.
'SHERRY' UPDATED.
'SHERRY' COPIED.
```

Sample CREATE-ACCOUNT Usage.

DELETE-ACCOUNT deletes an account and all its files from an PICK system.

DELETE-ACCOUNT is a PROC which runs the program DEL-ACC in SYSPROG-PL. The PICK/BASIC program removes the SYSTEM D-pointer for the account, and puts it in SYSPROG's MD. Then it removes all D-pointers to data files from all the dictionaries on that account and places them in the accounts MD. the program then calls on the DELETE-FILE verb, which deletes the account's MD, plus all dictionary and data-level files for that account, from SYSPROG's MD.

Requirements to run DELETE-ACCOUNT:

1. You must be logged on to SYSPROG.

2. SYSPROG must have Q-pointers to the MD of the account, and to SYSTEM.

3. D-items must exist in DICT SYSTEM for SYSPROG and the account name.

4. SYSPROG must have access to SYSTEM and all files on the account to be deleted.

ALL USERS SHOULD LOG OFF before running this because an item in the SYSTEM dictionary will be deleted.

The DEL-ACC program produces a listing of all files being deleted.

```
>DELETE-ACCOUNT                                    PROC name is typed

ACCOUNT NAME ?SHERRY                               at TCL.

FILES TO BE DELETED IN ACCOUNT SHERRY     02 APR 78      PAGE   1

FILE            BASE    MOD  SEP

MD              34593   37   1
GEN/LED         85344   1    1
GEN/LED         49911   231  1
BP              44319   17   5

 DO YOU STILL WANT TO DELETE THE ACCOUNT ?YES     Must start with 'Y'
```

Sample DELETE-ACCOUNT usage.

10.22 FILE STATISTICS REPORT

The File Statistics Report is a valuable tool for data base management. This Report is automatically generated by running a FILE-SAVE, or may be generated at any time by using the PROC LIST-FILE-STATS.

The FILE-SAVE process creates one item in the STAT-FILE for each D-pointer saved on the file-save tape. a listing of the STAT-FILE is created at the end of every file-save. the same listing can be generated from TCL by the LIST-FILE-STATS PROC.

The statistics report adds data security by providing a list of file Base, Modulo and Separation parameters, and by recording the order of files on a FILE-SAVE tape.

The report is broken down by account, with a line of information generated for each file in the account that includes:

total and average item size
total and average number of items per group
utilization of file-space
actual data stored, and "pad" space used in the file

A total line is generated for each account showing the total:

items
bytes (characters)
frames (includes linked)
group format errors

Creation of the STAT-FILE dictionary and data areas is part of the SYS-GEN procedure. STAT-FILE is contained on the System Programmer (SYSPROG) Account. As it is normally updated from this account, there is no need for STAT-FILE on any other account.

Alternately, the file may be created via the following:

```
CREATE-FILE (STAT-FILE 1,3 29,1)  [CR]
```

When a FILE-SAVE is started, the STAT-FILE data area is cleared and the current file statistics information is written into the data area.

The STAT-FILE data area will also be empty after a file-restore is done, be cause Attribute 1 of the file definition is a DY. This is desirable as the statistics are no longer applicable.

It is helpful to make synonym accounts in the SYSTEM dictionary Q-pointers, so that there is only one D-pointer for each account. this way, the data for the account will be saved under the one account name that is a D item in SYSTEM.

The item-id in the STAT-FILE is of the form:

t:n

where "t" is the tape reel number where the file was dumped (this will be 0 if the SAVE was run without dumping data to the tape); and "n" is the FILE-NUMBER. This file-number is used in the selective restoration of files using "SEL-RESTORE".

Note that files in the listing that have a SIZE field of zero are synonym D-pointer files, that is, a previously found D-pointer caused the data to in the file to be dumped.

The NAME field of the items in the STAT-FILE contains data in the form:

accountname*dictname*dataname

where one, two or all three of the fields may be present, depending on whether the file is an account, a dictionary, or a data-file.

| This topic describes a number of special utility verbs. |

STRIP-SOURCE Verb

The STRIP-SOURCE verb is a TCL-II verb used to remove the source code from Assembly Language programs. this frees large amounts of disc space back to the available space pool. modes with source stripped out out can still be verified against the ABS.

FORMAT:

STRIP-SOURCE file-name item-list

After the verb has been invoked, the user is prompted with:

DESTINATION FILE:

The file-name where the stripped object code is to be stored should then be entered. For example:

```
>STRIP-SOURCE PROG * [CR]
DESTINATION FILE-SPROG [CR]
```

Here the file PROG containing source programs is stripped and copied to the file SPROG.

The first six lines of the source item will be copied without source code stripping. Standard Pick Systems convention for source modes has the "FRAME" statement in line 1, and other descriptive information in lines 2 through 6; this information is maintained through the STRIP-SOURCE process.

LOCK-FRAME Verb

The LOCK-FRAME verb may be used to core lock a frame.

FORMAT:

LOCK-FRAME number

where "number" is a decimal frame number. The LOCK-FRAME verb responds with the absolute hexadecimal work address of the memory buffer in which the frame is corelocked. The frame remains corelocked until it is released by the UNLOCK-FRAME verb, or the system is re-booted.

UNLOCK-FRAME Verb

The UNLOCK-FRAME verb clears the corelocked buffer status of the frame indicated.

FORMAT:

UNLOCK-FRAME number

where "number" is a decimal frame number.

CHARGES Verb

The CHARGES verb prints the current computer usage since logon as connect time in minutes and CPU usage in charge-units.

FORMAT:

CHARGES

CHARGE-TO Verb

The CHARGE-TO verb is used to keep track of computer usage for several projects associated with the same logon name.

FORMAT:

CHARGE-TO name

This verb performs the following:

1. Terminates the current charge session by updating the ACC file with the user's accumulated charge-units, line printer pages and connect-time statistics.

2. Changes the logon name to the original name concatenated with an asterisk and then the name following "CHARGE-TO".

For example, if the user is currently logged on to SYSPROG, and he types in the following:

```
>CHARGE-TO PROJECT1 [CR]
```

the LOGON name in the ACC file for the process will be changed to "SYSPROG*PROJECT1".

10.24 SYS-GEN AND FILE-SAVE TAPES: FORMAT

System restore is the process of "bringing up", or creating, the PICK system. a bootstrap program, all system software and all files can be loaded from magnetic tape. The system configuration is set up at cold-start time.

A system can be restored from a SYS-GEN tape or a file-save tape. there are three sections on a SYS-GEN tape:

1. The bootstrap section contains the MONITOR, the configurator, and some virtual program frames needed to bootstrap the system. There are 33 tape records in this section, followed by an END-OF-FILE mark (EOF).

2. The ABS section, which contains the system software. this section is preceeded by a tape label, which contains the release level, and terminated by an EOF. this software makes up the PICK Operating System, the Language processors (ACCESS, PICK/BASIC, PROC, ASSEMBLY), and the various utility programs.

3. The FILES section contains a minimum set of PICK files, including the SYSTEM dictionary, a SYSPROG account, and the POINTER-FILE, SYS-ERRS, ERRMSG and ACC files. each account is preceeded by a tape label containing the account name, and is followed by an EOF. the last account on the tape is followed by two (2) EOF's (called an EOD, or END-OF-DATA mark).

A FILE-SAVE tape contains only the third section--Files. There are no coldstart nor ABS sections on FILE-SAVE tapes, only files.

File-restores cause files to be created, loaded and integrated into the system.

A file-restore can be initiated from a bootstrap, by use of the F option.

Sequence of Events in File-restores

The first event in a complete file-restore is the initialization of available overflow space to the complete range on the system from the process workspaces (WSSTART) forward to the end of disc (MAXFID).

The SYSTEM dictionary is then created and cleared. Then the first account Master Dictionary (MD) is created and a pointer to it is placed in the SYSTEM dictionary. Then the first file dictionary is created and a pointer to it is placed in the account's Master Dictionary. Next is the data file, which will proceed in one of two ways:

1. The slow method. The file is created, a pointer is added to the dictionary, and then the data is loaded. This method is necessary if reallocation is being done, or if the file is the POINTER-FILE.

2. The fast method. The file is loaded group by group as it is created. After it is completely loaded, a pointer is placed in the dictionary. This is the normal method.

Next, the file dictionary is loaded. The next file's dictionary and data sections are created and loaded, and so forth until all of the accounts are finished, when the SYSTEM dictionary is loaded. This completes the file-restore.

Account-restores proceed in the same sequence, except that the SYSTEM Dictionary is already present, and only the pointer to the account Master Dictionary is added to it.

Console Listing Accompanying File-restore

The figure below is an example of a file-restore listing. Each line corresponds to a file pointer. Each line is indented in accordance with the level of the file in which the pointer is placed. The file name is first followed by the base, modulo, and separation of the file as it is being restored. An (S) following the line indicates that the pointer has the same base as some other pointer already listed and that that file has already been created.

TERMINAL RESPONSE FOR ADDITIONAL REELS

When a tape reaches the end-of-tape mark without having finished the routine which it is executing, it will send the "mount next tape" message. When the next tape is mounted, the process will wait for the character 'C', at which time it will check the tape label on the new tape for admissability. If it does not like the tape label, it will say so, and wait for the tape to be changed to the correct tape, and the character 'C' to be entered.

It may occur that the tape that was mounted, which had a label that the processor did not like, was the correct tape, however. It is now possible to execute the response 'O', for override. This will cause the tape accept the new reel without continuing to complain about the label. It may also cause the processor to complain about the label on the next tape, at which time the use of the 'O' response is recommended.

```
>SEL-RESTORE TEST * (N) [CR]          Fake SEL-RESTORE to look at tape

FILE#:999[CR]                         Non-existant file number

SYSTEM 1                              SYSTEM dictionary pointer
 BLOCK-CONVERT 2                      BLOCK-CONVERT dictionary
  BLOCK-CONVERT (S)                   DATA section same as DICT section
 SYSPROG 4                            SYSTEM pointer to SYSPROG account
  SM 5                                dictionary
   SM 6                               data
   SM77 7                             another data file
    .
    .                                 files in SYSPROG account
    .
 MD (S)                               SYSPROG'S MD
 M/DICT (S)
 SHERRY 65                            new account
  INVOICE 66                          Shared dictionary
   INVOICE-FEB 67                     Multiple DATA pointers
   INVOICE-JAN 68
   INVOICE-MAR 69
    .
    .                                 Other accounts
    .
 ACC 144                              SYSTEM pointer to ACC account
  ACC 145                             DICT ACC file in ACC account
   ACC 146                            DATA ACC file
```

Sample FILE-RESTORE Console Listing.

If parity errors or other errors mar the files section of a FILE-SAVE tape, some data may be lost. The file-restore will continue, but operator assistance may be needed.

Parity Error Recovery Procedure

If a parity error is detected on a file restore, the prompt:

```
PARITY ERROR!  ENTER A TO TRY AGAIN
I TO IGNORE?
```

will be printed. Entering 'I' will cause the data block to be accepted as it is from tape without data correction. The specific item and file affected cannot be determined except as can be judged by the tape position, and the current set of files which have not been completed.

Recovery From Destroyed Pointers

If tape information identifying a file is destroyed, it may be impossible for the restore to create that file and subsequent files in the right order. the message:

```
ERROR IN DSEGMENT
@fff.ddd
LEVEL (1-3)?'
```

will be printed. fff.ddd gives the frame and hex displacement of the software location at which the error was detected. The operator is expected to advise the restore processor whether to:

1. Search for and continue with the next account on tape,

2. Search for the next dictionary file on tape, or

3. Search for the next data file on tape.

The response requires the operator's judgment as to the positioning of files on the tape and the total situation.

10.27 SELECTIVE RESTORES

The Selective-Restore capability allows individual files or items to be loaded onto an PICK system from a file-save tape.

This verb is used to restore a file from a system or account file save tape.

Selective restores are performed as follows:

1. Log on to the account with the file to be restored.

2. Mount the tape.

NOTE: Selective-restores may be started from any tape of a multi-tape file-save! To save time in searching a tape, the STAT-FILE listing may be consulted to determine which reel the file's data starts on, and that reel may be mounted. a SEL-RESTORE may be started at any place on any reel of the file-save tape. any coldstart or ABS sections will be skipped automatically.

3. Attach the tape unit (T-ATT)

4. To start the restore, enter:

```
>SEL-RESTORE file-name {item-list} {(options)} [CR]
```

where "file-name" is the file in which items will be placed. This file must be defined on the account from which the restore is run. The optional item-list enumerates those items eligible for restore. '*' may be used to indicate all items on the tape.

The data may be restored from either a specific file-name on the tape, or a file-number; the file-number may be obtained from a listing of the STAT-FILE when the tape was created.

If the N option is NOT used, the operator will be prompted:

```
ACCOUNT NAME ON TAPE?account-name

FILE NAME?file-name
```

where 'account-name' is the name of the account under which the file was saved on tape, and 'file-name' is the name of the file as it appears on the tape. Entering [CR] to 'FILE NAME?' causes the account Master Dictionary (MD) to be restored. The file-name may be of the form file-name, DICT file-name, of file-name,data-name.

If the N option is used, the prompt will be:

FILE #?

and the file-number must then be entered.

As the tape is searched, the file-names on it are printed, along with the file-numbers; names are indented one space for account-names, two spaces for dictionaries, and three for data-file-names.

Applicable options are:

O Overlay items already on the file.

A The tape is already positioned in the desired account. In this case the "ACCOUNT NAME ON TAPE" prompt will not appear.

N The file is to be identified on tape by its file number, in which case the prompt will be FILE#?. The required file # is the one which accompanies the file on the statistics file print-out for the appropriate file save.

I The item-ids of the restored items will not be printed.

C This option has effect when the 'N' option is used. It causes every item before the next end of file to be a candidate for restore. This ensures that data can be restored even if a D pointer is damaged on the tape.

S Skips forward spacing of the tape. this is used when restoring from 2.4 or 2.5 tapes, or when at the beginning of the second or later reels of a file-save.

Hints:

If a STAT-FILE listing for the tape is available, ensure that the account-name and file-name are on the tape as you think they should be. In the case of multiple D-pointers in the SYSTEM dictionary to an account, or multiple D-pointers in the M/DICT to the file, the account-name or file-name on the tape will be the first one the save processor encounters, and may be different from the one commonly used by you. All other names will appear in the STAT-FILE listing with no data (null SIZE field), and cannot be specified in the SEL-RESTORE!

If in doubt about the contents of the tape, the files can be listed by using a SEL-RESTORE of the form:

```
>SEL-RESTORE TEMP *

ACCOUNT-NAME ON TAPE?  XXXXX

FILE-NAME?  YYYYY
```

where XXXXX and YYYYY are fake names that will cause the SEL-RESTORE to search the tape for non-existent data; files will be printed out as encountered, along with the file-numbers. Files with an (S) are synonyms, and should be ignored.

In restoring both the dictionary and data section of a file, restore the dictionary first (DICT filename). remember that the dictionary items FOLLOW the data items, so for large files, there may be a considerable pause after the time that the system has found the file (it stops the printout), and the actual restore of the items.

At any point, the tape may be backed up (T-BCK (n)), or forward-spaced (T-FWD (n)) to position it, and a SEL-RESTORE with the A or N options may be started; this may be faster than restarting the tape from the beginning when restoring both the dictionary and the data sections of a file, or when restoring multiple files.

Remember also that account dictionaries (M/DICT items) FOLLOW ALL OTHER FILES for the account on the tape.

To restore the Q-pointers in the SYSTEM dictionary, use the N option with FILE# = 1. Remember that this will be the last file on the tape! On a multi-reel file-save, mount reel #1 first, and start the SEL-RESTORE as usual; when the file-name "SYSTEM" has printed out, use the BREAK key to interrupt the restore, then mount the LAST reel of the set, and type "G[CR]" to continue the process. This saves searching the entire first reel and any intermediate reels of tape.

The PICK system has the ability to save the entire disk data base on magnetic tape and to restore the tape copy, entirely or selectively, to disc. It is this procedure that provides backup in the event of a catastrophic failure or error.

IT IS YOUR RESPONSIBLITY TO DO SAVES FREQUENTLY ENOUGH TO ENSURE ADEQUATE BACKUP FOR YOUR PARTICULAR SITUATION!

The FILE-SAVE procedure protects your valuable data base by creating an off-line copy of it on magnetic tape. Tape is a inexpensive commodity when compared to the time and effort invested in your data base. It is vital that you protect that investment through adequate backup. As a MINIMUM pratice you should have separate daily backup tape-sets for one week's time and a monthly backup for each month in the previous year. Some situations may also need a weekly backup cycle for the past month. That is, use a separate tape-set for each day of the week, one for each week of the month and one for each month of the year. The longer cycle tape-sets should be stored off premises to provide protection in the event of physical damage such as fire.

ONLY YOU CAN DETERMINE WHAT IS ADEQUATE FOR THE PROTECTION OF YOUR DATA!

The FILE-SAVE procedure is quite easy. You simply mount the tape reel onto which you intend to save your data, and then LOGON to the FILE-SAVE account. The FILE-SAVE will ask a few simple questions and then will begin saving your data. When it has finished it will rewind the tape and log itself OFF. Operator intervention is required only if the data to be saved exceeds one tape reel (approximately 28 Mb. per 2400-ft. reel). You then have a complete backup of your disk data base.

The FILE-SAVE procedure normally creates a list on the terminal of the files it finds as it saves the data base. It will output error messages if it encounters unusual or illegal conditions but it will attempt to continue to save data. If the terminal you run the save on is not a hard copy terminal, you may want to send the listing to your printer. You can do so by answering "Y" to the first question "CONSOLE LIST TO PRINTER?".

The FILE-SAVE generates statistics of the saved data as a by-product. Answer "Y" to the question "DO YOU WANT FILE STAT REPORT?"if you desire a printed report displaying the current file statistics.

SAVE is the verb that performs a FILE-SAVE; it is called by the FILE-SAVE PROC.

The FILE-SAVE PROC sets up a sentence using the SAVE verb.

FORMAT:

SAVE {(options)}

OPTION	MEANING
D	Data area is saved. this option must be present if any Files are to be saved.
F	File names are printed. if (F) is not specified, just the SYSTEM file and account-names are listed.
G	Group Format Errors are repaired. GFE's are also logged in the STAT-FILE, if the (S) option is present.
M	Bit Map of all FID's of frames in Files on the system is generated. (Not implemented fully).
N	No overflow space is required to run the SAVE. this makes it possible to perform a FILE-SAVE on a system that has no overflow space available. NOTE: if there are more than 1500 Files on the system, one (1) frame of overflow space will be needed for every 125 files above 1500.
P	Output (list of file names) goes to the line printer. if (P) is not specified, all output goes to the user's terminal.
S	STAT-FILE items are stored, one for each file saved. must be present if a STAT-FILE listing is made after the FILE-SAVE.
T	Output to Magnetic Tape. if the (T) option is not specified, nothing will be written on magnetic tape. however, the STAT-FILE will be generated if the (S) option is used.

Files whose file definition items have a "DX" in line 1 will not be saved. thus any data file, dictionary or even an entire account may be prevented from taking up space on the FILE-SAVE tape.

Files whose file definition items have a "DY" in line 1 will be saved, but none of the items in the file or sub-files will be saved. the data section of the STAT-FILE, for instance, has a "DY" code, because the data is not valid after a file-restore, and needs not be saved.

To prevent spurious (fake) Group Format Error messages from occurring on other lines while the FILE-SAVE is running, the SAVE processor locks groups as it saves them. up to 4 groups may be locked at one time by a file-save process. these groups would be the ones containing:

1. The SYSTEM dictionary pointer for the account being saved.

2. The file dictionary pointer for the dictionary of the file being saved. this would be a group in the account's MD.

3. The group in the data file being saved.

4. A group in the dictionary of the ACC file.

If a user on another line tries to access data in a locked group, his terminal will hang until the file-save process finishes saving all the items in that group and unlocks it.

If the (T) option is specified, the SAVE processor will prompt the user's terminal:

FILE-SAVE TAPE LABEL =

The response will be written on the tape as part of the tape label

10.29.1 MULTIPLE REEL SAVES

When the data to be saved exceeds the capacity of the mounted reel, a MOUNT NEXT REEL message will appear on the terminal screen. The cursor will be prompting the operator for input, immediately following a pound-sign (#) which has been displayed as part of the MOUNT NEXT REEL message.

Remove the tape reel, which should have rewound itself. Mount and position the next reel to the BOT mark. (Beginning Of Tape) Make sure you have inserted a write-ring. Make sure tape drive is on-line. Now enter the appropriate character at the #.

C - CONTINUE

O - OVERWRITE (used in cases of erroneous tape labels)

Q - QUIT

This procedure also holds true for RESTORING multiple reels.

The system has the ability to save and restore single accounts. The ACCOUNT-RESTORE verb is used to add a single account to an already existing PICK system. The ACCOUNT-SAVE PROC allows you to generate a save tape with only one account on it.

ACCOUNT-RESTORE

An Account-restore can be performed from a save of a whole system or from an Account-save tape. in either case, the account will be restored and a pointer to the account will be created in the SYSTEM dictionary. NOTE - the account must not already exist on the system. Account-restores may be started from any tape of a multi-tape file-save! To save time in searching a tape, the STAT-FILE listing may be consulted to determine which reel the account's data starts on, and that reel may be mounted.

Account restores are performed as follows:

1. Log on to SYSPROG
2. Mount the tape with the account on it.
3. Type: ACCOUNT-RESTORE new-account-name [CR]
   ACCOUNT NAME ON TAPE? old-account-name [CR]

The operator must respond with the name of the account, the same name under which the save tape saved the account must be used. The tape will be searched for the account, and the restore will proceed automatically.

A 'Synonym' segment may be encountered with a base which has not been found on the tape. This would happen if a D pointer on the saved account pointed to a file on another account, or if a 'D' segment on the tape was unrecognizable because of a parity error. In this case, the message 'SYNONYM NOT FOUND' will appear. The synonym D-pointer will not be created but the restore will continue.

ACCOUNT-SAVE PROC

The 'ACCOUNT-SAVE' PROC functions similarly to the 'FILE-SAVE' PROC. The files section contains no System Dictionary pointer or items, and only one account is saved. No synonym D or Q pointers will be saved. If STAT-FILE items are generated, they will pertain only to the saved account.

Account saves are performed as follows:

1. Log onto SYSPRO
2. Mount a tape with a write ring.
3. Type: ACCOUNT-SAVE [CR]
   TAPE LABEL IF DESIRED tape-label [CR]
   ACCOUNT NAME? account-to-be-saved [CR]

The response to this must be an account name in the system dictionary.

The WHAT verb is used to display the system configuration, the current status of its locks and tables, and the location of the processes that are logged on. The WHERE verb is a subset of the WHAT verb.

FORMAT:

WHAT {(options)}

where the options may be "P" for output to the printer, and "S" for generating the "WHERE"-data (see below) in numerical sequence.

FORMAT:

WHERE {n} {(options)}

WHERE may be used to display data for all channels that are logged on; if the optional "n" is used, only data for channel n is displayed. The where verb also allows specification of a range of lines as well as the specification of an account name. The default form of the WHERE displays all lines which are logged-on. Display of the status of lines not logged-on by the where verb requires the use of the 'Z' option.

The WHAT verb has several selectable parts. The system configuration is displayed in every case. The option 'L' will suppress the display of the locks; the option 'W' will suppress the display of the WHERE component; and the 'S' option will suppress the display of the SP-STATUS component. If a numeric or a numeric range is included, it will be applied the the WHERE component, as will an account name specification. The 'Z' option will apply to the WHERE component, and the 'A' option will apply to the SP-STATUS component. Some examples follow:

WHERE 3-5	Displays the return stack for users three through five.
WHERE 'DP'	Displays the return stack for all lines logged onto DP.
WHAT L	Will suppress the locks section of the WHAT verb.
WHAT W	Will suppress the WHERE section of the WHAT verb.
WHAT S	Will suppress the SP-STATUS section of the WHAT verb.
WHAT LWS	Will yeild only the system configuration section of the WHAT verb.
WHAT 'account-name'	Will display only those lines which have the account account-name logged onto them.

New forms of the WHO, WHAT, and WHERE verbs.

The WHAT verb displays the state of the system as below: (numbers in brackets are not part of display).

```
15:03:22  19 JAN 1978
CORE LINES PCB0 WSSTART WSSIZE SYSBASE/MOD/SEP  MAXFID DEBASE AVAILSPACE
 64K   17   512   1056    100     6156  11   1  97799       0    51234
       [1]  [2]    [3]    [4]      ...[5].....  [6]       [7]      [8]

 15679 (3D3F)-11                                                    [9]

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                     [10]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                     [10]
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                     [10]

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                     [11]
00 00 00 00 00 00 00 00 00 00 00 00

 16 0400 3F 00   165.083 164.04C
*02 0240 7F 00   121.000 121.059
 00 0200 7B 00     6.15B 183.123 181.07F
 11 0360 5F 00    11.156  16.0F4
 01 0220 7D 40    21.033   6.070   6.033  16.1C8  13.042
 07 02E0 7D 00  T  6.070   6.033   5.054
 08 0300 7D 00     6.070   6.033   5.054
[12][13][14][15][16]....[17].................................

SPOOLER IS NOT ACTIVE
LINE PRINTER READY.
```

NOTES:

[1] Number of communication lines (terminals) plus one (spooler) = number of processes on system.

[2] PCB-FID for channel zero; each following channel's PCB-FID is displaced by 32 frames from PCB0.

[3] Extended work-space starting FID; WSSTART = PCB0 + 32*LINES (Including SPOOLER).

[4] Extended work-space size; number of frames per workspace is either 100 or 66 frames. There are 3 workspaces per line.

[5] System base-FID/modulo/separation; SYSBASE = WSSTART + WSSIZE*3*LINES.

[6] Maximum disc FID; (T50 - 97799; T200 - 364999).

[7] Reserved.

[8] Available overflow space; linked frames + contiguous frames.

[9] Group-locks (if any); format= ddddd (xxxxx)-cc where:

ddddd=group FID (decimal); xxxxx=group FID (hex); cc=channel number

channel number is internally coded from PCB-FID of channel; upper nybble of lower byte concatenated with lower nybble of upper byte; it displays as the actual channel number, however.

[10] PICK/BASIC locks (48); system reserved (15); spooler-linking-work-space (last bit). Bits start @ 127.20.
[11] System lock bytes; 00=available; else has channel # as above.

LOCK#	LOC	USAGE
0	127.0	Lock-table lock.
1	127.1	Overflow table lock.
2	127.2	Group-lock table lock.
3	127.3	MESSAGE processor lock.
4-29		Reserved.

Data equivalent to "WHERE" follows.

Sequence of channels is in current priority chain sequence, except for those channels that have a PIB-status of "7D" (waiting for terminal input), which are not in the chain and therefore appear in numerical sequence. If the "S" option is used in the WHAT verb, all channels are in numerical sequence.

[12] Channel number; preceded by a "*" if your channel.
[13] PCB-FID (hex) of channel.
[14] PIB-status of channel; 7F/FF = Active, or ready to go.
7B/FB = Terminal output; 7D = Terminal input;
5F = Waiting for disc; 3F = Release Quantum/Sleeping.
Typically spooler is "3F".
[15] PIB-status-2; 00 = Normal; 40 = In DEBUGGER.
[16] "T"= Tape attached; "P"= Printer attached.
[17] Location counter (first address) & subroutine return=stack addresses.

Entry format = fff.lll where fff = decimal FID; lll = hex. location.

Typical locations:
5 = TCL
6/9 = Terminal I/O;
13-16 = EDITOR;
21 = DEBUGGER;
22-32 = ASSEMBLER
53-70 = ACCESS Compiler;
71-100 = LIST;
107,109,180-189= PICK/BASIC;
190-199= PICK/BASIC Compl
290-298= RUNOFF;
165= SPOOLER.
Note: in above data, channels 7 & 8 are at TCL prompt; channel 1 is in the debugger.

| The VERIFY-SYSTEM PROC checks to see if the system software is correct. |

The VERIFY-SYTEM PROC generates a check-sum for every frame of software, from 1 to 399. These check-sums are compared with those in the ERRMSG file, in an item named "CHECK-SUM". This item contains the correct check-sum for all the system software frames. Each line in the item contains a check-sum for one frame of code, optionally followed by one or more hexadecimal limits. If the limits are present, the check-sum is generated only for bytes within the limits. If no limits are present, the check-sum is generated for bytes 0--X'1FF'. This is done because some frames contain tables which change from time to time, such as the system overflow table. Table entries are not check-summed, only assembly code.

If all the program frames verify, error message 341 is printed:

[341] RELEASE XX.X SYSTEM VERIFIED.

If a frame generates a check-sum which does not match the check-sum for that frame in the "CHECK-SUM" item, the FID of the frame, the generated check-sum and the stored check-sum from the item are printed, and error message 342 is printed at the end of the check run:

```
[342] RELEASE XX.X SYSTEM DOES NOT VERIFY!
THERE ARE n PROGRAM FRAMES WITH MISMATCHES!
```

Where n is the number of programs whose check-sums do not match.

The VERIFY-SYSTEM PROC should be run whenever it is suspected that the system software is in error.

If a mismatch is found, the software can be restored by mounting the sys-gen tape and using the 'A' boot option.

10.33 USER-DEFINED CURSOR CONTROL IN PICK/BASIC

PICK PICK/BASIC supports ADDS REGENT 100 terminals; to use other manufacturer's terminals, the user must supply his own system software.

The cursor control (@) function in PICK/BASIC is designed to run on a REGENT terminal. To do this, set the terminal type to R. (See the section on the TERM verb.)

To use another terminal, pick a terminal type other than R. When you use the @ function in PICK/BASIC, the PICK/BASIC run-time package performs a subroutine call to frame 399 (SYSTEM-CURSOR.) The interface is:

T0	contains the X co-ordinate.
CTR11	also contains the X co-ordinate.
CTR10	contains the Y co-ordinate, if there is one. otherwise, CTR10 will be negative.
R15	points one byte before the location of the cursor control string to be generated by the user's assembly program.

NOTE: Cursor control strings are built directly in the PICK/BASIC program's descriptor table, using a direct string (X'02') descriptor type. Cursor control strings cannot exceed 8 bytes in length! If you generate a cursor control string of more than 8 bytes, you will destroy the PICK/BASIC program's descriptor table!

The user should look at the item SYSTEM-CURSOR in the USER-MODES file for examples of how to program cursor control for different terminals.

SECTION 11

ICON/PICK TERMINAL DEVICES

ICON®

11.1 Terminal Control Facilities: An Introduction

Most modern application programs take advantage of the enhanced characteristics offered by asynchronous terminals. Features such as reverse video and row/column addressing are common, as well as a host of other enhanced characteristics that vary from vendor to vendor. An installed site also has a vast plethora of terminals to chose from, each offering a feature or capability that make them unique in the marketplace. In order to cope with the reality of options given to the buyer, ICON/PICK has facilities to upgrade current terminal definitions to incorporate new features, as well as define entirely new cursor types to the system. It is the purpose of this chapter to describe in detail the issue of system terminal control.

CONVENTIONS USED IN THIS CHAPTER

CONVENTION	MEANING
UPPER CASE	Characters printed in upper case are required and must appear exactly as shown.
lower case	Characters printed in lower case are parameters provided by the user.
[ ]	Braces surrounding a parameter indicate that the parameter is optional
TERMTYPE	The single character code that defines a port as a type of terminal, the same character that shows in the TERM TYPE field of the TERM verb.

11.2 EDIT-TERMINALS

Terminal definitions proc, allowing a system administrator to alter or define new terminals and make them available to system users.

FORMAT:

EDIT-TERMINALS

EDIT-TERMINALS PROC

The EDIT-TERMINALS PROC will modify or generate terminal definitions according to specifications provided by the user in a menu driven environment. If new terminal definitions are to be generated, the user must have a manufacturer's technical reference for that terminal. This PROC is only available to the SYSPROG account.

After entering the proc name EDIT-TERMINALS at TCL, the following menu will be displayed:

```
Current terminal definitions are

*W   wyse50     *V   viewpoint    *D   dt1200     L   LSI
v    vt100      *A   ansi         t    tvi970     T   tvi940
I    IBMPC      *H   hazeltine    *R   regent

1) Activate a terminal definition
2) Deactivate a terminal definition
3) Modify/Define a terminal definition
4) Delete a terminal definition
5) Copy an EXISTING terminal definition

Enter selection, RETURN to exit
```

Beneath the heading *Current terminal definitions are* is the list of terminals that may be understood by the ICON/PICK system. The * character before a terminal name indicates that this terminal type has been loaded by this PROC or by the LOAD-TERMINALS verb and is currently available to users on the system. The single character preceding the long version of the name is the TERMTYPE code used when setting a port to a specific TERMTYPE via the TERM verb. The rest of the entry is the long version of the terminal name. This is also the item name in the TERMINALS file, typically found in the SHARED-FILES account.

11.2.1 Activate a Terminal Definition

Activating a terminal definition is the functional equivalent of executing the LOAD-TERMINALS PROC. An additional prompt is displayed

```
Enter terminal name (* means ALL)
```

By entering the * character, all of the terminal types will be loaded and available for use by the TERM verb. The user may optionally enter the long name of the terminal, after which only that terminal type will be loaded. Upon completion, the screen will be repainted and the main menu displayed again. Note that the TERMTYPE character must be unique for each terminal definition loaded. If there are duplicates, terminal definitions should be loaded selectively one at a time.

11.2.2 Deactivate a Terminal Definition

Deactivating a terminal definition is the functional equivalent of executing the CLEAR-TERMINALS verb. An additional prompt is displayed:

```
Enter terminal name (* means ALL)
```

By entering the * character, all of the terminal types will be cleared, leaving no system ability to do any special terminal codes. The user may optionally enter the long name of the terminal to be deactivated. This is particularly useful if the definition of a terminal has been changed and needs to be reloaded, as an active terminal cannot be reloaded.

11.2.3 Modify/Define a Terminal Definition

Modifying or Defining a terminal definition is the most powerful feature of this proc. The ICON/PICK system is shipped with a variety of terminals, but it is this feature that allows a vendor or a user site to customize an application package by exploiting the features of a terminal.

When using this option, there are a number of prompts that must be responded to, typically with the character code for a given function. These codes can be entered in one of three ways; by entering the ASCII mnemonic, by the decimal or hexadecimal value, or by an ASCII character.

EXAMPLE: ENTERING A TERMINAL CODE SEQUENCE	
CODE:	Escape "A" char(9)
HEXIDECIMAL:	.1B .41 .09
DECIMAL:	27 65 9
MNEMONIC:	ESC A HT

Notice that hexadecimal numbers are preceded by a period. The mnemonic codes are the typical ASCII codes for the characters, which can be found in Section 9.134, *List of ASCII Codes*, in the ICON/PICK User Manual (P/N 171-010-001).

The first prompt asks for the terminal type, which is the long name of the terminal. If this is a Modification, then an item with the terminal name exists in the TERMINALS file and will be retrieved at this time. As the user progresses through the various definition items, the values found in the item will appear as defaults, allowing the user to press the return key if no modification is required. Default values will appear within brackets in the ASCII mnemonic form (using the example above, the default would appear as [ESC A HT]).

If a terminal definition is being created (which is to say the value entered is not an item in the TERMINALS file), the message New terminal type will appear.

The next prompt will request the TERMINAL TYPE code, which is the TERMTYPE code.

The Cursor address type is the next prompt. This requests the type of row/column addressing codes that are used. The large majority of terminals use one of four types of addressing schemes, one of which must be entered here:

1- Adds Regent
2 - Televideo
3 - ANSI
4 - WYSE 132

For example, the Televideo type of addressing is done with four characters; an ESC, followed by an "=", followed by row, and then column, each occupying a byte. If it is not altogether clear from the technical documentation what type of scheme a given terminal is using, a little experimentation with this prompt may yield results.

Lead escape sequence is not currently in use.

Number of entries refers to the number of characteristics for a given terminal to be maintained by ICON/PICK in the system tables. Any terminal will have at least ten attributes, and the system will currently support up to 28. This number reflects how many prompts will be displayed, as the PROC will not ask for features that are not to be included in the definition. Null attributes are allowable.

Size of each entry will alert the system to the longest possible sequence of characters to be used. Four is typical, although there are terminals that have functions requiring as many as eight.

The remainder of the prompts will request specific codes, as can be seen in the following example. By using the terminal manufacturer's technical reference guide, a terminal definition may be constructed in a short time.

EXAMPLE: Modify/Define a terminal definition

```
Terminal type            :
Terminal code            :
Cursor address type      :
1 - Adds Regent, 2 - Televideo, 3 - ANSI, 4 - WYSE 132

Lead escape sequence     :
Number of entries        :
Size of each entry       :
Clear screen             :
Home cursor              :
Clear to end of screen   :
Clear to end of line     :
Start blink              :
Stop blink               :
Enable protect mode      :
Disable protect mode     :
Cursor back              :
Cursor up                :
Cursor right             :
Cursor down              :
Start reverse video      :
Stop reverse video       :
Start underline          :
Stop underline           :
Start protect            :
Stop protect             :
Enable transparent       :
Disable transparent      :
Lock keyboard            :
Unlock keyboard          :
Insert line              :
Delete line              :
Insert character         :
Delete character         :
```

11.2.4 Delete a Terminal Definition

Deleting a terminal definition is functionally equivalent to removing an item from the TERMINALS file. This function will completely erase a specific terminal definition.

11.2.5 Copy an Existing Terminal Definition

Copying an existing terminal definition is the functional equivalent of copying one item of the TERMINALS file to another, with the addition of requesting the user to enter a new TERMTYPE code. This is especially useful for making new definitions, as you can copy the definition of an existing terminal to a new item and have most of the work taken care of.

NOTE: It is impossible to copy a definition to itself.

11.3 CLEAR-TERMINALS

The CLEAR-TERMINALS verb allows a user to disable a terminal definition system wide.

FORMAT:

CLEAR-TERMINALS termtypes

where *termtypes* is the single character TERMTYPE that has been previously loaded into the system by the LOAD-TERMINALS verb. The TERMTYPE displayed when the TERM verb is executed with no options.

11.4 LIST-TERMINALS

The LIST-TERMINALS verb displays all active TERM types and the corresponding terminal name.

FORMAT:

LIST-TERMINALS [(S)]

This verb will list all active TERM types, which are activated through the EDIT-TERMINALS utility or LOAD-TERMINALS verb. Displayed are the one character TERM codes, followed by the terminal name. A * appears next to the current TERM type for the current line. If the *S* option is used, only the active TERM characters are displayed.

EXAMPLE:

>LIST-TERMINALS

Available terminal type characters are :

S	SOROC	t	TV910	y	VT100	Y	vt52
R	regent	w	wyse50	d	Dialog	B	bantam
*D	dt1200	P	pertec	L	lsi		

The one character TERM TYPE is displayed, followed by the terminal name. A * precedes the current TERM TYPE of this port.

>LIST-TERMINALS (S)

Available terminal type characters are : StyYRwdBDPL

The one character TERM TYPE codes are displayed.

11.5 LOAD-TERMINALS

Load a terminal definition item from a file into the system so that a new TERM TYPE is available to the users.

FORMAT:

LOAD-TERMINALS filename [itemname] [*]

The item loaded from the file must be in the proper format, having been created by the EDIT-TERMINALS PROC.

EXAMPLE:

>LOAD-TERMINALS TERMINALS DT1200

Will load the terminal definition for the DT1200 into the system terminal tables.

11.6 TERM

Terminal and/or line printer characteristics may be displayed or set by a process via the TERM command.

FORMAT:

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

ARGUMENTS:

a is the terminal line length (i.e.,number of characters per line). The **a** parameter must be in the following range: 16<**a**<140.

b is the number of print lines per page on the terminal.

c is the number of blank lines per page on the terminal (sum of **b** and **c** equals page length).

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

e is the number of delay characters following each top-of-form. If **e** is zero, no form-feed character will be sent to either the terminal or the printer. If **e** is non-zero, a form-feed character is also output before each page; if **e** is 1, this character is sent to the line-printer, but not to the terminal. If **e** is greater than 1, the form-feed character is also sent to the terminal at the beginning of each page and that many delay or idle characters is also sent to allow the terminal time to settle after the form-feed. The form-feed character sent to the printer is always a hexadecimal '0C' (ASCII FF character).

f is the backspace character. An ASCII backspace (control-H) is always input to backspace over (or erase) the last character that was input; however, the user may set the actual character echoed to his terminal. This accommodates terminals that cannot physically backspace, or that have a backspace character other than the ASCII backspace. The **f** parameter should be 21 for the ADDS REGENT terminal, and 8 for the TEC 2402 terminal.

g is the line printer line length.

h is the line printer page length.

t is the terminal type code. This changes the form-feed character sent by the system to match the terminal requirements and, more importantly, sets the appropriate cursor addressing for the BASIC cursor functions. A few TERMTYPES are:

A - ADDS 580
D - DIALOG
L - LEAR-SIEGLER ADM-3A
d - ICON DT1200
R - ADDS REGENT
T - TV950
V - ADDS VIEWPOINT
X - NO CURSOR ADDRESSING FUNCTIONS

Individual parameters may be null; i.e., as 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.

EXAMPLE:

>TERM <CR>

	Terminal	Printer
PAGE WIDTH:	79	132
PAGE DEPTH:	24	64
LINE SKIP :	0	
LF DELAY :	1	
FF DELAY :	1	
BACKSPACE :	21	
TERM TYPE :	R	

Standard terminal characteristics set for the ADDS REGENT terminal.

>TERM ,,,,2 <CR>

Resets the FF delay to 2 in order to get a clear-screen on the terminal.

>TERM <CR>

	Terminal	Printer
PAGE WIDTH:	79	132
PAGE DEPTH:	24	64
LINE SKIP :	0	
LF DELAY :	1	
FF DELAY :	2	
BACKSPACE :	21	

>TERM ,,,,,,120,48 <CR>

Resets the line-printer page size to 120x48.

>TERM <CR>

	Terminal	Printer
PAGE WIDTH:	79	120
PAGE DEPTH:	24	48
LINE SKIP :	0	
LF DELAY :	1	
FF DELAY :	2	
BACKSPACE :	21	

Use the TERM or SET-TERM verb to allow the ICON/PICK system to echo the proper control characters for a given terminal type. The TERM verb will reset the TERMINAL TYPE until the user executes the OFF verb. The SET-TERM verb will reset the TERMINAL TYPE until the users executes another SET-TERM or TERM verb.

EXAMPLE:

>TERM w

Will set the TERMINAL TYPE to "w" until the user executes the OFF verb.

>SET-TERM w

Will set the TERMINAL TYPE to "w".

SECTION 12

ICON/PICK UNIX PROCESSOR

ICON®

12.1 Introduction to ICON/PICK/UNIX Processor

The ICON/PICK/UNIX Processor (IPU processor) is a product of the unique concurrent operating systems technology offered by ICON. This chapter discusses the utilities that provide the user a "window" into the ICON/UX systems, both to execute commands and to transfer data.

There are two basic forms to the IPU processor functions; the first is an extension of the TCL processor, and the second is used by invoking new TCL verbs. There are two basic types of commands; those that execute commands in another operating system, and those that transfer data. This chapter will discuss the new extensions to ICON/PICK and then conclude with a technical discussion on the impact and limitations of the extensions.

12.2 EXPORT

The export command allows data to be transferred from the ICON/PICK database file system into the ICON/UX file system on a item by item basis.

FORMAT:

EXPORT filename

This will cause another prompt to appear:

Enter the Item ID:

which is a request for an item name. The prompt

Enter the UNIX file name

requests a path name for an ICON/UX file. No assumptions can be made as to the directory the file will be placed in if a full path name is not specified. All ICON/UX file names should be specified with a full path name.

Conversions will automatically be performed when data is transferred. All attribute marks (char(254)) will be converted to new-line (char(10)) characters when **EXPORT**ing. No conversions are done on value marks or subvalue marks. The conversion facilitates ease of using ICON/UX utilities, such as the vi editor or the itroff text formatter.

The **EXPORT** verb as of **ICON/PICK** Release 3.00 does not process LISTs, but requires a one by one transfer of items. This feature will be present in a future release.

EXAMPLE:

```
>EXPORT BP
Enter the Item ID: MYPORG
Enter the UNIX file name: /usr/max/basic/myprog.b

Take the item MYPROG in the BP file and copy it into myprog.b

found in the /usr/max/basic directory.
```

12.3 IMPORT

The **IMPORT** command is used to copy data from the ICON/UX file system a single file at a time. Each ICON/UX file is placed within an item of a file, with appropriate conversions being performed.

FORMAT:

IMPORT filename itemname

After the **IMPORT** command is executed, a prompt will appear:

Enter the UNIX filename:

after which the user must enter the full pathname of the ICON/UX file. It is important to remember that the ICON/PICK system does not have a locality of reference in the ICON/UX tree structured file system; the system does not depend on any particular directory for operation.

12.4 UX Verb

The UX command will execute an ICON/UX command and display the results on the users terminal, or alternatively execute a shell specified in the ETC.FILE of the SYSPROG account.

FORMAT:

UX {command} {[opt1] [opt2] [opt3]...[optn]}

the UX command will execute a given ICON/UX command and display the results on the users terminal. For example:

UX vi /usr/tom/myprog

will invoke the ICON/UX vi full screen editor and operate on the file myprog found in the /usr/tom directory.

EXAMPLE:

UX ls /usr/don	list the files of the /usr/don director
UX	execute the shell found in the ETC.FILE
UX uniplex	invoke the uniplex application program

Note that features normally expected by ICON/UX shells such as output redirection and pipes are not concurrently supported when executing ICON/UX commands from the ICON/PICK operating system.

12.5 PASSWD

The PASSWD proc will enable an account to use the facilities of the ICON/UX systems.

FORMAT:

PASSWD

The PASSWD proc is available in the SYSPROG account for the system administrator to enable ICON/UX privileges on an account basis.

EXAMPLE:

```
>PASSWD

1 root      2 daemon    3 uucp      4 sys
5 don       6 tom       7 ellen     8 rjt
9 remo      10 jan      11 kay      12 skl

Enter the name of the ICON/PICK account: SYSPROG
Enter the number of the ICON/UNIX login: 5
Give dos privileges?                   : N
```

The first question should be answered with a name of a valid account within the system; SYSPROG is the account entered above.

The second prompt requires the user to enter the number of one if the ICON/PICK logins displayed above; 5 is entered above to give the SYSPROG account don privileges when executing ICON/UX commands.

Finally, DOS capabilities can be enabled here autonomously from ICON/UX capability. This option will work only if the hardware option is present in the machine, and has been properly installed and configured.

NOTE:

System Orientation: a difference in Operating system philosophy.

There are two distinct approaches to system organization that are worth mentioning here in an effort to allow the casual user an understanding of both the ICON/PICK and ICON/UX systems.

The PICK system security and orientation is defined by the account. Accounts tends to be logically organized within a given data domain; for example, in a typical organization all of the payroll programs and data may be found within an account named PAYROLL, all of the general ledger data in an account named GENLEG, etc. Accounts tend to be autonomous from other accounts. A user is typically given the password(s) to the account(s) that they are required to interact with. This tends to cause certain users who perform a variety of functions to have several passwords (this is especially true where each account needs to maintain a certain amount of security, causing synonym file pointers to be unacceptable). An important distinction of the orientation here is that where the account exists as a functional domain, several users may hold passwords to the account and use it on a regular basis.

The UNIX system security and orientation tends to be defined by the file, and hence in a beta definition by the user login (the owner of the file). Accounts tend to be personal with each user of the system having their own account and set of files. Application programs can be made available to users either globally or in groups. Data domains tend to be the directory, a repository for a logically related data set or collection of files.

The bridge between the systems is typically executed with a single UNIX login available per account.

When using the IPU processor, the user is causing the ICON/PICK system to relinquish control and allow one of the ICON/UX systems to execute a command or shell of some kind. Logically, the ICON/PICK process will be put to sleep and control given to the ICON/UX system for the duration of the command executed. The ICON/UX systems are true operating systems and perform many of the same functions an ICON/PICK user would normally have available at TCL. In addition, there are other facilities such as on-line manual helps, full screen editors, various compilers and an assembler, as well as text formatting facilities.

12.6 The ! Command Line Processor

Removed from Pick - use UX

The ! command line processor provides a shorthand version of the UX command for user.

FORMAT:

! command {[options]}

By preceding an ICON/UX command with an ! (exclamation mark) at TCL, privileged users may execute ICON/UX commands. This form may not be used from within a PROC or a BASIC EXECUTE statement.

EXAMPLE:

! ls/usr	List the files of the /usr directory
! man pipe	Display online documentation for pipes
! id	Display effective user and group id
! WHO	Returns to TCL and WHO is not an ICON/UX command

COMMENTS

ICON/PICK USER'S MANUAL **P/N 172-026-001**

Your comments and suggestions are appreciated and will help us to provide you with the very best in system and application documentation. Please send your comments to the address at the bottom of this page.

1. How would you rate this manual for COMPLETENESS? (Please Circle)
   Excellent Poor
   5 ~~~~~~~~~~~~ 4 ~~~~~~~~~~~~ 3 ~~~~~~~~~~~~ 2 ~~~~~~~~~~~~ 1 ~~~~~~~~~~~~ 0

2. Is there any information that you feel should be included or removed?

3. How would you rate this manual for ACCURACY? (Please Circle)
   Excellent Poor
   5 ~~~~~~~~~~~~ 4 ~~~~~~~~~~~~ 3 ~~~~~~~~~~~~ 2 ~~~~~~~~~~~~ 1 ~~~~~~~~~~~~ 0

4. Indicate the page number and nature of any error(s) found in this manual.

5. How would you rate this manual for USABILITY? (Please Circle)
   Excellent Poor
   5 ~~~~~~~~~~~~ 4 ~~~~~~~~~~~~ 3 ~~~~~~~~~~~~ 2 ~~~~~~~~~~~~ 1 ~~~~~~~~~~~~ 0

6. Describe any format or packaging problems you have experienced with this manual and/or binder.

7. Do you have any general comments or suggestions regarding this publication or future publications?

Your Name _____

Company _____

Address _____ Phone (___)_____

City & State _____ Zip Code _____

Job Function _____

Type of Equipment Installed: _____

Printed in the U.S.A. Rev. A 172-026-001