Malcolm Bull
Training and Consultancy Publications

MB-Guide to

The Pick System

Malcolm Bull

MB-Guide

to

The Pick system

MB-Guide

to

The Pick system

by

Malcolm Bull

Malcolm Bull
Training and Consultancy Publications
19 Smith House Lane
BRIGHOUSE
HD6 2JY
West Yorkshire
United Kingdom

Telephone: 0484-713577

ISBN: 1 873283 73 3

Edition: 2.5
Updated: 24:09:93

P r e f a c e

The *MB-Guide to the Pick system* looks at the essential features of the Pick system.

This *MB-Guide* discusses:

* A general introduction to the Pick system.

* The basic hardware associated with the system.

* The standard software which accompanies the system: Access, the Pick enquiry language; TCL and the TCL stacker facilities; the Editor; the Basic language; Procs; Runoff; Spooler.

* Files: the logical and physical organisation of accounts, files and items.

The material will be of interest to anyone who is moving to a Pick system, whether they are completely new to computing or migrating to Pick from some other computing system.

You will find the the *MB-Guide* beginner's guide series and the *MB-Master* self-tuition courses of interest in conjunction with the material presented in this *MB-Guide*.

You may find the following titles in the *MB-Guide* beginner's guide series useful in conjunction with the present volume:

Advanced Pick
Operations & system management
Files: sizing & monitoring
Using the Pick Editor

You may find the following *MB-Master* self-tuition courses of interest in conjunction with the material presented in this *MB-Guide*:

PICK1: Starting Advanced Pick
PICK2: Pick systems management
PICK3: Running your Pick system

This *MB-Guide* is not intended to present an exhaustive description of the subject but merely to place it in context and give the reader enough information to use the facilities and to survive.

Best use can be made of this *MB-Guide* if it is read in conjunction with the reference literature which is provided for your system. You should amend your copy of this guide so that it accurately reflects the situation and the commands which are used on the implementation which you are using. By doing this, your *MB-Guide* will become a working document that you can use in your daily work.

I hope that you enjoy reading and using this *MB-Guide* and the others in the series, and welcome your comments.

C o n t e n t s

Introduction to Pick

The software project which was to become known as *the Pick system* was conceived in the mid-1960s by Dick Pick and Don Nelson working in the USA. The first implementation of the Pick system appeared around 1974 when Pick and Microdata (later to become McDonnell Douglas) worked together to produce a commercial database management system based upon Microdata hardware; this was known as the *Reality* system. Pick parted from Microdata and went on to produce other versions of the Pick system in collaboration with other manufacturers and vendors. Since that time, the system has gone from strength to strength and is available on a wide range of equipment, from a small single-user PC implementation up to large systems - such as those used by local government and police authorities - supporting several hundred users. Today, you may encounter Pick and Pick-like systems in many guises. The name of the system may be different:

Pick / Open Architecture / Advanced Pick
Reality
Prime/Information
Revelation or Advanced Revelation
Ultimate
Unidata
Universe

but whatever the nature of the hardware and whatever implementation you are using, the underlying database model is the same as that developed by Pick and Nelson in the 1960's, and with very little effort (and virtually no un-learning) a programmer can move from one system to another. In many cases, the software, too, is portable between the various systems.

The main features of the Pick system include:

* On-line creation and maintenance of data files.

* Multi-user support, allowing many users to work on the system at the same time.

* Simultaneous access to the same files by several users.

* Unlimited file size.

* Flexible field and record format.

* Spooler software enabling many users to produce printed output on one (or many) printers.

* Reading / writing data to magnetic tape / diskette devices.

* The Access enquiry language which allows users to produce their own reports by means of data dictionaries created for the files.

* The Basic language allowing users to write their own

processing routines.

and a great many more which we shall look at in this *MB-Guide.*

The Pick system is one of the most powerful means of performing commercial computing, and with its Access enquiry language it represents a database management system whose power is almost unchallenged.

Like all computer systems, your Pick system comprises two parts: the hardware and the software.

1.1 R83

The most popular implementation of Pick is that known as R83, release 83. Since its original launch in the early 1980s, this release has gone through several versions and the current version is R83 version 3.1. It is this release which is the basis for the present *MB-Guide.*

All other implementations of the system (with the exception of any mentioned below) are based upon the R83 release. In most cases, the individual licensees have added their own facilities and use their own version numbering such as R91, and these are largely for cosmetic ends, providing front-end menus by which the users can call up the standard features of the system.

1.2 Advanced Pick

After the development of the R83 release, Pick Systems made a number of radical changes to the system:

* File indexing
* Macros
* Menus
* A TCL stacker to store and re-issue TCL commands.

So great were these changes, that they represented more than just another version of R83. Instead this version, originally to be called R84, was named OA, Open Architecture.

Further changes were made to the underlying strategy of R83 and OA and this version, known as AP, Advanced Pick, was announced during 1988.

The major additional features in Advanced Pick were:

* A Update Processor to facilitate the creating and maintenance of items and files by way of the dictionary definitions.

* An Output processor based upon Runoff and the Update Processor, allowing the user to produce documents and other textual material.

* Additional processing codes to ensure file integrity and the ability to pass from one file to another during file processing with the Update processor.

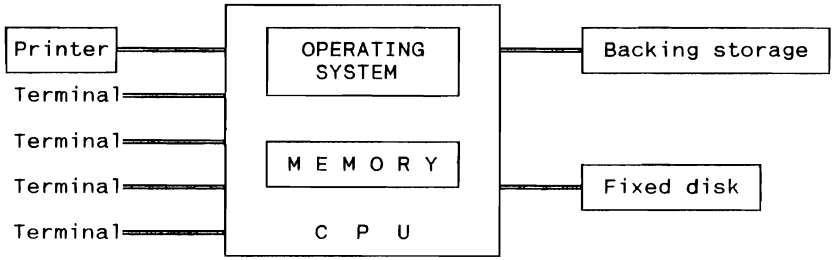
* An ability to interact with other systems, such as DOS and Unix.

Pick Systems major products are now the R83 release and the AP release.

This subject is covered separately in the *MB-Guides to Advanced Pick: AP/DOS and AP/NATIVE.*

2 Hardware

The Pick system is available on a wide range of equipment. Some systems are based on personal computers which serve only one user, others support several users, whilst others will handle many hundreds of terminals.



Whatever its size, every Pick system is made up of the same basic components:

1) The central processor unit and the system itself.

2) The memory: this holds all the data and programs which are being processed by the various users at any moment.

3) The user terminals: Pick is designed to be used by several users at one time, each user working from his/her own terminal. Each terminal is connected to the computer via a connection known as a *port*. The actual number of ports depends upon the particular configuration of the system which you are using. Pick will support a wide range of terminal equipment, including VDUs and printer-terminals, but many features of the system are designed specifically for VDU terminals and screen displays.

4) The printer: any number of printers may be connected to the system, and these may be small serial printers or full-size line-printers. A part of the system software, known as the *spooler*, enables two or more users to produce printed reports at the same time.

5) The backing storage: this is used to pass data from one system to another and to produce back-up copies of the system and its files for security purposes. The backing storage device may be a standard reel-to-reel magnetic tape deck, or a tape-streamer, or a floppy-disk.

6) The fixed disk: all the programs and the data files created and processed by the individual users and those used by the system itself are held permanently on a fixed disk. Disk storage space is shared by all users of the system, and the number of disks and the size of the available disk space depends upon the configuration of your particular system. The *virtual memory* feature of Pick means that this disk space can be regarded as an extension of the memory space.

Based around this fundamental configuration, it is perfectly

feasible to attach other equipment and devices such as bar-code readers and encoders, plotters, point-of-sales terminals and fax and telex machines.

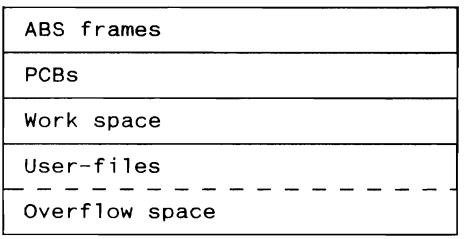
A number of commands are available for interrogating the details about your system. We shall look at these briefly below. These and other commands are described in the *MB-Guide to operations and systems management.*

Let's look more closely at the hardware attached to your Pick system.

2.1 Hard disk

The most important part of the hardware on the Pick system is the hard disk. The hard disk is used as an extension of core (or RAM or memory), allowing the users to access any part of the database as and when required. This access mechanism is controlled by the virtual memory feature of the system.

The storage space on the hard disk is allocated for use by the system itself, and for general file storage by the users:



* ABS frames: this is the area in which the programs for all the standard Pick software are held. Special PICKWARE products, such as CompuSheet+ and Accu/Plot and Jet, also use these frames for their own special routines, and a number of frames are set aside for users to load their own assembly language processing routines. The ABS section normally occupies about 1024 frames.

* PCBs: the process control blocks for each available port (plus one for the spooler) are held here. These hold registers and other control information relating to each individual process. Each PCB is 32 frames in size.

* Work space: this is an area of virtual memory which is used as an extension of the PCB as a work area and holds the information relating to each process. There is one workspace for each available port (plus one for the spooler). Each Workspace is 381 frames (that is, 3 times 127 frames) long. On McDonnell Douglas implementations, each user has 18 frames, unless additional workspace has been assigned in attribute 8 of the account-definition item for that account.

* User-files: finally comes that area of disk which holds the data files used by the systems and the users. The SYSTEM file is normally the first file in this area followed by the MD for the accounts and their files. The start of this area is known by the symbolic name SYSBASE.

* Overflow space: beyond the area occupied by the files comes the unused area of disk which will be used whenever new files are created and when an existing file requires more frames to accommodate additional items and as files and items grow in size.

The extent of each area may vary according to the implementation which you are using. There is no fixed division between the user-files area and the overflow space. When a file-restore operation has been performed, the disk is fairly tidy with all the user-files grouped together and one large chunk of overflow space following. However, in time, the user-files take over parts of the overflow space as new files are created and as existing files expand, and overflow frames occur in the user-file space as existing files and items are deleted.

A number of commands are available for monitoring disk usage and the fragmentation of the disk space, and for interrogating other details about your system. These are described in the *MB-Guide to operations and systems management.*

The WHAT command will display details about your system:

```
CORE LINES PCB0 WSSTART WSSIZE SYSBASE/MOD/SEP MAXFID AVAIL OVERFLOW
636K     7  704     928    127    3595  11   1 75582       23982

 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

 00 00 00 00 00 00 00 00 00 00

*00 02C0 FF30     121.000 121.1AB 166.602
 06 0380 BF30     170.229 170.147

 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.
 The SPOOLER is in an unambiguous state.
```

The POVF command displays figures for the available disk space, and indicates the fragmentation of the available space. A typical display might look like this:

```
 6934-  6934 :      1   12466- 12514 :     49
12577- 12732 :    156   12907- 13275 :    369
32658- 33627 :    970   45324- 46916 :   1593
57898- 75582 : 17685

TOTAL NUMBER OF CONTIGUOUS FRAMES     : 20823
```

When creating a new file, the system will scan the above table for the smallest *gap* into which the file can be placed and then allocate these frames to the file and remove them from the table.

If the available disk space becomes too fragmented, it may prevent the creation of new files and may make overflow of existing files inefficient. In such circumstances, a file-save and file-restore procedure should be carried out.

2.2 Virtual memory

The Pick system uses a virtual memory facility to render all data (which is permanently stored on hard disk) available in memory when needed by the users. Frames of data are continually being read into memory and held there for inspection, amendment and deletion by the users. The frames are written back to disk when memory becomes filled with write-required buffers space must be made for new frames of data to be brought into memory, and also when the system is closed down.

In the worst case, the entire contents of memory may be frames which have been updated in memory but not yet written back to disk. For this reason, the machine must be properly shutdown, as described later, and not just switched off, otherwise the data in memory will be inconsistent with the data on disk.

Some implementations allow you to control the manner in which updated buffers are written back to disk, so as to minimise the danger of losing your data.

Those who are using the system are placed in a queue, first come, first served; the person at the head of the queue is allowed to use the system when it is next available. Typically, each user gets a time-slice of 10 milliseconds in which to carry out some of his/her work. This is the maximum time which is allowed for each process before it is interrupted and placed at the end of a queue of all the processes waiting to run. A process may be interrupted and placed in the queue in other circumstances:

* It may have requested data which has to be read from disk. The process can only continue when the required data is in memory.

* It may have attempted to read a frame which is currently being updated.

* It may have issued a SLEEP (or RQM) command to

relinquish its time-slice.

* It may have executed a command which expects the user to enter data at the keyboard. The process can only continue when the user has entered the data.

Some implementations allow you to change the duration of the time-slice.

2.3 Terminals

A set of parameters describes each terminal - the terminal characteristics - and the operating uses these parameters to control all input/output to the terminal, to determine the format (width and depth) of all output reports and also the terminal display protocol. If a terminal does not have the correct terminal type and other characteristics, then the cursor control, and screen display and formatting features may not work correctly.

The terminal characteristics can be displayed by means of the TERM command. A typical display from this command might look like this:

Each terminal is associated with a set of parameters - the terminal characteristics - which the operating uses to control all input/output to the terminal. To determine the format (width and depth) of all output reports and also the terminal display protocol.

The terminal characteristics can be displayed by means of the command:

TERM

A typical display from this command might look like this:

```
            TERMINAL PRINTER
PAGE WIDTH:     79      132
PAGE DEPTH:     24       64
LINE SKIP :      0
LF DELAY  :      2
FF DELAY  :      2
BACKSPACE :      8
TERM TYPE :      I
```

These parameters are used by all the standard Pick processors and changing any of the parameters may affect the output produced.

+ Page width: the number of characters per line for the device.

+ Page depth: the number of lines for the device

+ Line skip: the number of lines to be left at the foot of the screen. This is the difference between then *page depth* value for the screen and the physical depth of

the screen.

+ Line-feed delay: the number of delay characters which are to be output after each line-feed.

+ Form-feed delay: the number of delay characters which are to be output after each form-feed, or top-of-form skip.

+ Backspace: this is the decimal value of the character which the terminal is to interpret as the back-space character.

+ Terminal type: this is a code denoting the type of terminal which is being used. Some of these are shown below.

A ADDS	M Mime
A Adds 580	N Wyse 100
A CIE-ANT	N WYS WYSE100
B Ampex 210	P Hewlett-Packard 2621A
B Beehive	P Pertec 701
C C-Itoh VT52	Q QVT 102
C DTC	R Adds Regent
C VT52	R Regent
D Datamedia	S Soroc
D DEC VT100	S Wyse 60
E Emulog 200	T Mini-Tec 2401
F TV910	T Tec 2402
G GTC	T TV920
G IBM 3161	T TV950
H Honeywell VIP 7200	U Ultimate - Volker-Craig
I IBM monitor	V Adds Viewpoint
I MM-MON	V Ultimate - Viewpoint
J VT100	V Viewpoint
L Lear Seigler ADM-3A	W Wyse 50
L LSI	X Datagraphix
M Ampex D80	X no cursor control required

If a terminal does not have the correct terminal type, then the cursor control and formatting features will not work correctly.

Some implementations hold the terminal characteristics for each port on the item TERMTYPE on the ERRMSG file, others store them in items on the DICT section of the ACC file. When logging on, the terminal characteristics are reset to the default settings for that port.

The TERM command can be used to reset the terminal characteristics for the current port. To change the parameters for the *entire system*, the System Manager will use the SET-TERM command. Many implementations have a utility such as the DEFINE-TERMINAL, to set the characteristics for a specific terminal type.

The SET-PORT and the SET-BAUD commands may be available to set the communication characteristics and the baud rate for a

port. You are referred to your system Reference Manual for the format of this command on your implementation.

2.4 Printers

Two types of printer interface are used: *serial* and *parallel.* A printer can be attached to any suitable port on the Pick system. The STARTPTR command is used to declare the port a printer and to associate a specific form-queue(s) with the printer.

Slave printers can be attached to any port which is provided with an auxiliary output port. When the slave printer is activated, all output sent to the screen may also be sent to the slave printer. The appropriate terminal reference manual will give details of the control sequences which will activate and deactivate the slave printer.

2.5 Backing storage devices

In addition to the hard disk, the Pick system may be fitted with one or more of:

* 3.5" floppy diskette,
* 5.25" floppy diskette,
* 0.25" cartridge tape, and/or
* 0.5" reel-to-reel magnetic tape.

These are used to make copies of the contents of the hard disk, using the file-save and/or account-save features. They can also be used to archive data - such as last year's invoices - which are no longer required on the live system. We use the term *backing storage* to refer to these devices. There are a number of TCL commands available for controlling the backing storage device(s):

SET-8MM	T-BSR	T-RETEN
SET-DEVICE	T-CHK	T-REW
SET-FLOPPY	T-DET	T-SPACE
SET-HALF	T-EOD	T-STATUS
SET-SCT	T-ERASE	T-UNLD
SET-TAPE-TYPE	T-FSF	T-UNLOAD
SET.DT	T-FSR	T-VERIFY
SET.TM	T-FWD	T-WEOF
T-ATT	T-RDLBL	T-WTLBL
T-BCK	T-READ	
T-BSF	T-RET	

Further details on the use of these devices is given in the *MB-Guide to using backing storage.*

2.6 Back-up

We have already introduced the backing storage devices, and mentioned that their primary use is for making security

copies of the entire database. The operational aspects of the system, including file-save / file-restore and account-save / account-restore are covered in other volumes in the *MB-Guide* series:

File-save and file-restore
Files: monitoring and sizing
Operations and systems management
Security
The spooler

Here, we shall look at the general concepts.

2.7 File-save / restore

The entire data resources of your computer system reside on the hard disk - or disks - which your Pick system uses. It is obviously important that you make regular copies of the data on the disk, in case of hardware problems or in case someone accidentally deletes a file.

There are several ways in which data can be written to backing storage and recovered from backing storage. The diagram at the end of this section summarises these. In the following sections, we shall look at some of these facilities more closely. These activities are discussed in detail in the the *MB-Guide to File-save & file-restore.* There are a number of TCL commands and utilities associated with these operations:

:FILES
: restores the entire system data files from a file-save tape.

ABSDUMP
: dumps the ABS file to backing storage.

ACCOUNT-RESTORE
: restores a single account from a file-save tape or an account-save tape.

ACCOUNT-SAVE
: saves a single account on backing storage.

F-S
: an alternative to the FILE-SAVE utility.

FILE-SAVE
: saves the entire system on backing storage.

LIST-FILE-STATS
: output the file statistics currently held on the STAT-FILE.

RESTORE-ACCOUNTS
: uses a file-save tape to recover all accounts which are not currently on the system.

S-DUMP
: outputs the contents of selected items on a file sorted

according to any specified criteria.

SAVE
saves the entire system - ABS and/or data files - on backing storage. This is the fundamental TCL command which is harnessed by the ACCOUNT-SAVE, FILE-SAVE and other utilities.

SEL-RESTORE
restores one, several or all the items to a file from a file-save tape or from an account-save tape.

T-DUMP
dumps one, several or all the items on a file to magnetic tape.

T-LOAD
restores one, several or all the items on a file from a magnetic tape produced by the T-DUMP / ST-DUMP command.

VERIFY-SAVE
checks the integrity of a file-save or an account-save tape.

	To save	To recover
The system	File-save Binary-save	File-restore from file-save Binary-restore from binary-save
An account	Account-save	Account-restore from account-save or file-save
A file	T-DUMP file	T-LOAD file from T-DUMP Selective-restore from account-save or file-save
An item	T-DUMP file 'item'	T-LOAD file from T-DUMP Selective-restore from account-save or file-save

3 Software

Just as the logical organisation of the hardware of the the Pick system is essentially the same for all versions, so there is a range of software which comes as standard with the system.

Let's look more closely at the software which makes up your Pick system.

3.1 Access

Access is the Pick database enquiry language. It allows you and your users to make enquiries about your files and to produce displayed or printed reports based upon the contents of those files. These enquiries and reports are produced by typing English sentences such as:

```
LIST STOCK DESCRIPTION PRICE

SORT STOCK DESCRIPTION QTY PRICE WITH MATERIAL = "OAK"

SORT STAFF BY DEPARTMENT BY SALARY IF THE SEX = "F" AND
     THE AGE < "60"

SORT STOCK BY COLOUR DESCRIPTION TOTAL PRICE
```

Access is an example of a *query language* or an *enquiry language.* Access is a flexible and a powerful tool which enable non-technical users to get the best value from the data on their database. The language is so powerful that there is no longer any need to write report programs. Furthermore, Access is so simple that it can be used by non-technical users as well as by programmers and analysts.

The language is supported by a collection of field definitions which the analyst establishes on the *data dictionary* for each file. These definitions - for data elements such as DESCRIPTION, PRICE, QTY and MATERIAL in the above examples - can be used to extract and report any part of the database by means of an Access sentence.

We shall look at this topic more closely later in this *MB-Guide.* This subject is covered in detail in the *MB-Guide to Access* and in the reference literature for your system.

3.2 TCL

The terminal control language - TCL - is the most fundamental way of controlling and communicating with the Pick system. By typing in TCL commands, the user is able to perform a vast number of activities, including:

* Create and maintain data files and their contents.

* Create and maintain user-accounts.

* Copy data from one file to another.

* Create, maintain, compile and execute Basic language

programs.

* Execute standard programs and utility software.

* Create, maintain and invoke procedures written in the Proc language.

* Control and interrogate the usage made of the system.

* Control the spooler and the printers.

The TCL operations - the *verbs* - which are available are defined on a file called the *master dictionary* - the MD. Each separate account has its own MD. Every time a user types in a TCL command, the system searches through the MD to find the definition for the verb in the user's TCL command, and from this definition it determines what to do. If a particular account is not to be allowed to use any facility - such as the Editor - then the EDIT verb can be removed from the MD of that account. The verbs can also be *renamed* and you may even add your own programs to the MD so that they look and behave like TCL commands.

3.3 Editor

The Pick Editor is a piece of software which enables the users to create, change and delete the items on their files. It is a *line-editor* processing each line of the item in turn. Since it is rather a complicated tool, it is chiefly used by technical users for writing Basic programs and Procs.

Most systems offer much more user-friendly - and much more secure - means of maintaining their files. These may be specific user-application for handling particular files, or they may be general software such as Jet, or a similar screen editor, which displays the entire item which you are changing and allows you to move the cursor through the text, changing, deleting and adding text as you go.

These Editors are discussed in the *MB-Guide to the Editor* and in the reference literature for your system. We shall look at the Pick line-editor more closely later in this *MB-Guide.*

3.4 Basic

The dialect of Basic which is offered on the various implementations of the Pick system is a very powerful version of Dartmouth Basic. The language has been extended to allow programs to handle Pick disk data files and other aspects of the system. It has further features, including:

* Optional statement labels.

* Alphanumeric statement labels.

* Several statements on one line.

* Meaningful variable-names may be used, and variable-names may be of any length.

* Any variable may hold either numeric data or an alphanumeric string.

* Structured programming facilities: LOOP statement, CASE statement, IF ... THEN ... ELSE statement.

* Complex and multiline IF statements.

* Modular programming facilities: GOSUB statements for using internal subroutines, and CALL and SUBROUTINE statements for using external, independent subroutines are also available.

* External subroutines are written and developed as independent program units and may be used by one or more programs.

* Use of COMMON data to pass information between programs and external subroutines.

* File handling facilities: OPEN, READ, WRITE and DELETE statements.

* Locking to prevent several users updating the same record at the same time.

* Handling output on backing storage: magnetic tape, or floppy disk.

* Handling the data structure of Pick items: attributes, values and subvalues.

* Dynamic arrays: any number of elements, deletion and insertion of elements, sorting elements into sequence as they are added to the array.

* An interactive debugger to facilitate program testing and development.

* The power to invoke any TCL commands or Access sentences from within a program. The results may be returned for further use within the Basic program.

The language is also known by other names, including Data/Basic, Pick/Basic and Info/Basic. So powerful is the language that it really does belie its name.

This subject is covered in detail in the *MB-Guide to Basic,* in a number of associated guides and in the reference literature for your system. We shall look at this topic more closely later in this *MB-Guide.*

3.5 Procs

The Proc language was originally provided to allow a user to save a command or a sequence of several TCL commands as a single unit on the MD and then invoke this sequence - or procedure - as a single operation.

Since its inception, the Proc language has become very sophisticated and has now developed into a programming language in its own right: there are facilities for accepting keyboard input and for displaying output, for performing calculations and for using subroutines.

Procs are particularly useful when a complicated Access sentence is to be used many times, or when a complex sequence of operations is to be used by non-technical operators. By typing in the sentence and saving it as a Proc, it can be re-used many times simply by typing the name under which it has been saved.

This subject is covered in detail in the *MB-Guide to Creating and using Procs* and in the reference literature for your system. We shall look at this topic more closely later in this *MB-Guide.*

The Basic language has facilities to invoke any TCL command and Access sentence from within a Basic program. This means that Procs are gradually being replaced by more legible and more easily maintainable Basic programs.

There is also a Pick Assembler language. This is rarely used by commercial programmers, and is only of interest to those who are developing applications - or parts of applications - which need to utilise the low-level features of the computer.

3.6 Runoff

One of the fundamental components of Pick is the Runoff text processing feature. This offers many of the features of standard word processing systems: text input and output; text formatting; pagination with headings and footings; indexing; table of contents. Unfortunately, Runoff is not a *wysiwyg* - what you see is what you get - system. The text must be maintained by the Pick Editor and the format of the final document is only revealed when it is displayed or printed. For this reason, Runoff has been superseded by other proprietary software such as:

* Jet and The Works,

* Keyword

and other similar packages which offer many of the facilities which we have come to expect of a word processor. The Advanced Pick system offers the Output Processor as a standard text processing feature.

This subject is covered in detail in the *MB-Guide to Runoff* and in the *MB-Guide to the Output Processor* and in the reference literature for your system. We shall look at Runoff more closely later in this *MB-Guide.*

3.7 Spooler

Since Pick is a multiuser system and any or all of these users may be producing printed reports at the same time,

there must be some means of controlling the printers and the printed output. Printed output is monitored and controlled by a standard piece of Pick software known as the *spooler*. This offers facilities for:

* Simple jobs to be printed with no intervention from the user - the normal situation,

and also for:

* Supporting several printers.

* Producing multiple copies of reports.

* Killing and/or suppressing the output.

* Holding reports for inspection prior to printing and for overnight printing.

* Stationery alignment.

The ordinary user may work without any knowledge of the spooler if he/she does not require any of these special facilities.

This subject is covered in detail in the *MB-Guide to the spooler* and in the reference literature for your system. We shall look at this topic more closely later in this *MB-Guide*.

3.8 System generation tools

For users who want to develop their own application systems in a fourth generation language, rather than Basic, Pick is well supplied with system generation tools such as:

* System Builder / SB+
* Creator
* Libra

and many others. Typically, these systems allow the systems analyst / programmer to construct a Pick data dictionary and then use this as a basis for building:

* Data input / validation routines,

* File maintenance routines,

* Data enquiry screens,

* Displayed reports, and

* Printed reports.

This allows systems to be designed, developed, tested and implemented in a high-level medium, freeing the programmer from much of the routine of programming - opening files, reading records, updating files - and allows him/her to concentrate on the detail and the processing of the particular application system.

3.9 Other software

Since the Pick system was first implemented, many of its features have been overtaken by modern technology. As a result, Pick has shortcomings in certain areas:

* Graphics,
* Spreadsheets,
* Communications,

and in most cases these gaps have been filled by add-on software. To these, can be added an even wider range of applications software such as:

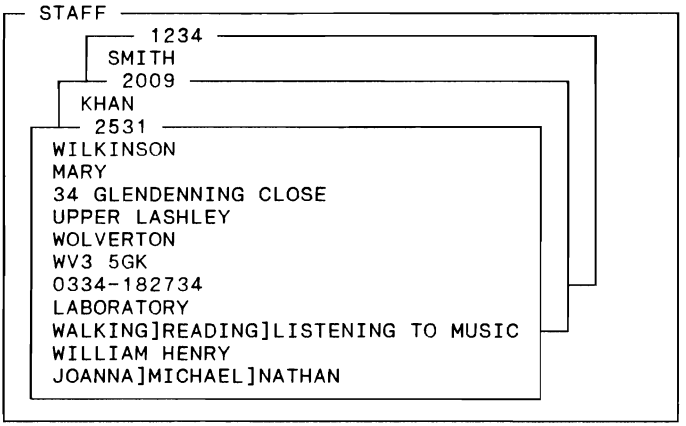
* Word processing,
* Stock control,
* Payroll processing,
* General ledgers and accounts,
* Financial and credit control,
* Vehicle scheduling,
* Catalogue, index and directory production,

allowing Pick to offer a range of commercial software which is virtually unrivalled by any other computer system.

Files

All the information which you process is organised as *files.* There may be a STAFF file to holds details of all the company's employees; a CLIENT file to holds details of all the company's customers, and many more. There may be any number of files on your system.

Each file comprises a number of separate *items* or *records.* Thus, the STAFF file may have one item for employee number 1234 (this may be John Smith), one for employee 2009 (Naren Khan), another for employee 2531 (Mary Wilkinson), and so on. There may be any number of items on a file. You can visualise a file like this:



Within each file, each item is identified by its *item-id* or *record key.* This is some unique identifier for that item. Typically, the items on the STAFF file may use the employee number as their item-id, so the item-id of Mary Wilkinson's item is 2531. Similarly, the items on the CLIENT file may use the account number of client reference number as their item-id.

Each item is made up of a number of data *attributes* or *fields.* Thus, the first attribute of a STAFF item may be the employee's surname; the second might be the employee's given names, and so on. There may be any number of attributes in an item. Normally, all the items on, say, the STAFF file will have the same number of attributes and these will always be in the same order. The nature, form and order of the attributes is determined by the systems analyst when he/she designs the file.

If a certain employee did not have, say, a telephone number, then this attribute (attribute 7 in our STAFF file shown here) would be empty. Some people call this a *null* attribute.

Some attributes, such as the hobbies (attribute 9) or the names of the children (attribute 11), may have more than one

value. Such an attribute is said to be a *multivalued* attribute. In the diagram, I have separated these *multivalues* by a square bracket, although your terminal may display some other character in place of the].

For security reasons, it is unwise to let every user have access to every file; otherwise, anyone could look at, and change, any record on, say, the STAFF file. Therefore, the files of your system are grouped into *accounts.* Thus, the STAFF file, the PAY.RATES file may be held on the WAGES account; the CLIENT file, ORDER file, PRICES file and the STOCK file may be held on the SALES account, and so on. To use the STAFF file, you must log on to the WAGES account, and to use the CLIENT file, you must log on to the SALES account.

4.1 File structure

A Pick *account* is a collection of files which are available to users of that account.

It is possible may log on to - and use - any account from any terminal, and there may be any number of people using an account at any time. In order to be able to log on to an account, a user must know - and enter correctly:

* The correct account-name - WAGES account or SALES account in our last illustration, and, if required,

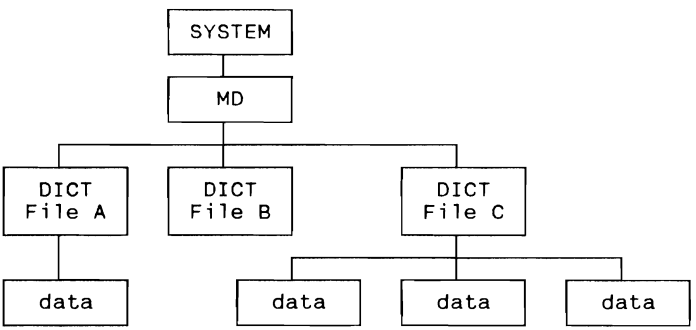
* The password for that account.

Without this information, it is not possible to log on to, and use, the Pick system.

Details of the accounts on your system are held on a file called SYSTEM, and your System Manager will maintain the accounts and the passwords of your system, as required.

4.2 SYSTEM, MD, DICT, data-file linkage

The file hierarchy is of fundamental importance in understanding the logical structure of the Pick system.



Each box in the diagram represents a *file,* and each line represents the *pointer-linkage* by which a record on one file

enables the system to locate a file *lower down* the diagram.

Let us have a look at the various components.

The file called SYSTEM is the most important file on any Pick system, and holds one record for every account on the system. There may be any number of accounts on your system. Each system has a number of standard accounts which hold general files and system administration programs used by the system manager in particular and all users. Important amongst these is the SYSPROG account which owns a number of files which are required by all users, including:

ACC
BLOCK-CONVERT
ERRMSG
POINTER-FILE
PROCLIB

which are made available to all other accounts via a set of *Q-pointers* on each MD.

The information in each record on the SYSTEM file points to the disk location of the *master dictionary* for that account. The master dictionary is known variously as MD and MD, and is a file containing definitions relating to all the files which you have created on that account, and definitions and pointers relating to all the standard system files. The MD also holds:

+ Definitions for all the standard TCL verbs,

+ Definitions for all the standard Access verbs and the modifiers and connectives,

+ Catalogued program pointers for all the standard system programs,

+ Catalogued program pointers for all those program which you have created and catalogued,

+ Pointers to all standard Procs,

+ Pointers to all the Procs which you have created.

On the MD, there is a record relating to each data-file, and this points to the dictionary section of that file, DICT, which holds the dictionary definitions which will allow you to use the Access language to interpret your data records.

In addition to the Access definitions, the DICT of the file also contains a record called the *data-level identifier* which points to the data section of that file.

According to the particular version of the Pick system which you are using, there may be only one data-level identifier and one data section - as shown by file A on the left of the diagram - or there may be several different data-level identifiers on the DICT of the file, each referring to one of a number of files which share the same dictionary - as shown

by file C on the right of the diagram.

The data-level identifier may have the same name as the file to which it refers, or on those implementations which only allow one data section, the data-level identifier may have the item-id DL/ID.

The data section of the file contains the true data records - our sales data, the personnel records, or the invoice information.

If the file has no data section - as with the file B in the centre of the diagram - then the data-level identifier on the DICT section points back to the DICT section. This probably means that you do not not want to use Access with the data which are held on this file. It may be a file used for holding programs or Runoff documents, for example. Such a file is often called a *DICT-only file*.

The file-definition items which relate to other files contain the following information about the file:

- The base fid = the frame identifier of the first frame in the file,
- The MOD = the modulo of the file,
- The SEP = the separation of the file.,
- The L/RET security = the lock-code to retrieve, or read from, the file,
- The L/UPD security = the lock-code to update, or write to, the file.

There are a couple of esoteric points which you might appreciate from the above diagram:

+ An account is any file for which there is a pointer on the SYSTEM file.
+ A master dictionary is any file for which there is a pointer on the SYSTEM file.
+ An account and its master dictionary are synonymous.
+ A file always has a pointer on the master dictionary of the account to which it belongs.
+ A file called PROGS held on the SALES account is quite separate from a file called PROGS held on the WAGES account.

4.3 File / item / item-id / data fields

The logical storage is accessed in terms of records on files.

* A system comprises a set of accounts.
* Each account owns a set of files.

* A file is a collection of records holding data of a similar nature. The records on the INVOICE file would hold information about invoices, the records on the PERSONNEL file would hold information about the staff and personnel, and so on. This is identical to the standard use of the term *file* within data processing.

* *item* is the Pick terminology for *record*.

* *item-id* is the Pick terminology for *record-key*.

* *attribute* is the Pick terminology for *field*.

* Each item may comprise one or more attributes separated by *attribute-marks*. This is ASCII character 254.

* Each attribute may comprise one or more *values* separated by *value-marks*. This is ASCII character 253.

* Each value may comprise one or more *subvalues* separated by *subvalue-marks*. This is ASCII character 252.

Some literature uses the term *secondary-value* instead of subvalue.

* Attributes, values, and subvalues are of variable length, each being terminated by the appropriate field-separator.

* The various fields - attributes, values, and subvalues - are identified by their sequential position within the item.

* Data fields - attributes, values, and subvalues - are held in variable-length format, each taking up as much space as necessary.

* Null (empty) fields are represented only by the associated (following) field-separator - attribute-mark, value-mark, or subvalue-mark.

Data design and record layout is as important in Pick as in any other system, and the structure of the data records would normally be decided by the analyst when the system was designed. The sort of reorganisation which we have considered here can, however, be implemented at any time, provided that the necessary changes are made to the Basic programs and Access dictionary definitions which process the file.

All physical storage space on the Pick system is allocated in terms of *frames*. A frame is normally 512 bytes, 1024 bytes or 2048 bytes in size. The actual size will depend upon your implementation. Use the DUMP verb:

```
DUMP 12345
```

to see the size of a frame.

A file is created as a set of an integral number of frames and as more data is added to the file, the size of the file increases in units of one frame.

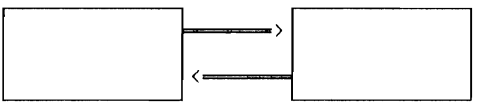
No frame is ever shared by two or more files.

MODEL 1: the smallest possible file, therefore, consists of one frame:

This is known as the *base frame* or the *primary file space.*

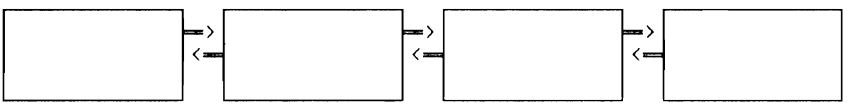
Only 500 bytes of each frame (or 1000 bytes or 2000 bytes in the case of the larger frames) are used to record the user's data. The remaining 12 (or 48 or 96) bytes of each frame are used by the system, as we shall see in a moment.

As records are added to the file and as the records themselves increase in size the data will take up more and more of the available space. Indeed, there may not be sufficient space in the single frame to accommodate the data. When this happens, the system looks around for an unused frame and writes the remainder of the data in this *overflow* frame, setting *pointers* or *links* from the base frame to the overflow and back. The overflow frame is almost certainly not the next frame on disk to the base frame. Someone else's process will almost certainly have used the next physical frame before you got a chance to need it.



The links which tell the system where to go next and to show where you have come from - are recorded in the first twelve bytes of each frame.

If the file grows further, then more and more frames will be tacked on to the end - each linking backwards and forwards.



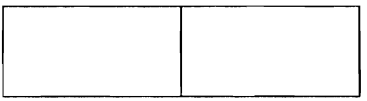
This is fine, but - because the system does not keep any sort of index to show which records are in which frame - each time you need to find a record, you have to start at the base frame and then read all the way through the intervening records until you find the one you need. So on average, you have to scan half the record on the file to find any specific record. If you are creating a *new* record, then you have to

read *all the way* through the file, just to be sure that the record doesn't already exist, and then to append the new record to the end of the file.

Each time the operating has to move its attention from one frame to another, it has to read the earlier frame, pick up the linkage information, move to the next overflow frame and so on until the frame holding the required item is found.

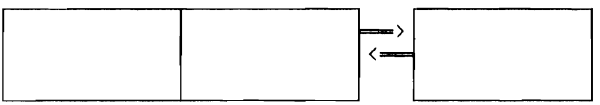
This is very time-consuming. The more records there are, the more time you consume.

MODEL 2: a first improvement which we might make to our simple model is to create our file as a *group* of, say, two contiguous frames:



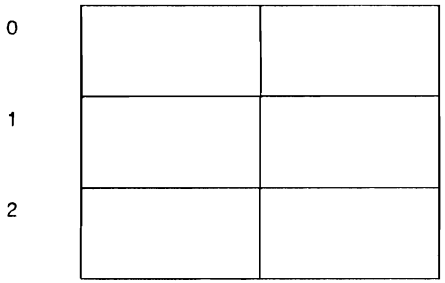
The first 12 bytes of each frame are still used to hold the link information, but the mechanical head which reads the frames will not have to travel so far. Indeed, some implementations with cache memory will read in a number of frames at one time, so if the following frame is required it is already available in cache memory without any need to read further frames from disk. This speeds up the rate at which frames can be processed.

As the file increases in size, any overflow frames are tacked on one at a time exactly as before:



In general, therefore, such a model of the file will speed up the physical access of the frames, but we still have to scan through the records to find the one we need.

MODEL 3: the third model of a file is one which consists not just of *one* such group of frames but *several* groups. In this example, we have three such groups, but you may have any number of groups.



The three groups are numbered (conventionally) 0, 1 and 2.

Now, when a record has to be written to the file, the system takes the record-key and performs an arithmetic calculation on it. The process is called *hashing*, as discussed later. The result of that calculation will be 0 or 1 or 2. According to this result, the record will then be written into that group. So each record goes uniquely into one of the groups.

When a record has to be read from the file, the same hashing calculation is performed and the appropriate group (0 or 1 or 2) is scanned for the required record, as it was in the earlier models.

All Pick files are structured in this way.

* Each file consists of a number of groups of frames.

* The number of groups is called the *modulo* of the file. This is often abbreviated to MOD. The first two files we looked at above had a modulo of 1. The last file had a modulo of 3.

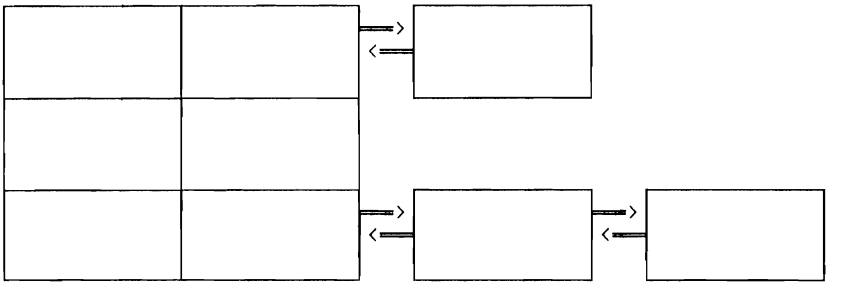
* Each group can consist of one or more frames.

* The number of frames in each group is called the *separation* of the file. This is often abbreviated to SEP.

* All the groups in a file have the same separation.

* Each group will overflow as necessary to accommodate the records which *hash* into that group.

Each group behaves rather like a separate physical file, and more overflow frames are tacked on to each group as number of records in the group grows.

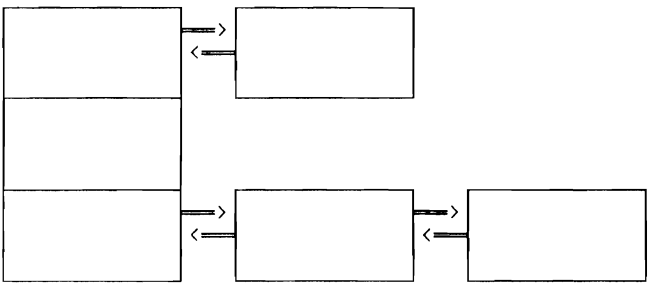


There are two further important points:

+ Because of the way in which the *hashing algorithm* works, it is best to have a *prime number* for the modulo. Otherwise, you may find *bunching* in your file, with some groups being completely empty whilst others are full to overflowing.

+ In practice, on those implementations which only read in one frame at a time, you lose the advantages of having a separation other than 1 - as justified when we derived our second model above. In such cases, it is much better to use a separation of 1 in all instances.

Having a separation of 1 reduces the amount of waste (unused) space in those groups which are not full.



This last example has a modulo of 3 and a separation of 1.

The modulo and separation are chosen when you create the file. In general, the larger the average size of the records, the larger the separation, and the greater the number of records on the file, the larger the modulo.

This topic is discussed further in the *MB-Guide to file design*.

4.4 Item format

Each frame of disk space consists of:

* The linkage bytes. These are the first 12 bytes of the

512-byte frame, or the first 24 bytes of the 1024-byte frame, and so on, depending upon the frame size used on your system.

* The data items.

  The general format of a physical item is:

```
xxxxiii^ddd^eee^fff^ggg^_
```

  where *xxxx* is a four-byte control field, *iii* is the item-id, *ddd*, *eee*, *fff* and so on are the data attributes of the item and the character shown here as ^ is the attribute-mark (hexadecimal FE) and the _ is the segment-mark (hexadecimal FF).

  In the case of the first item in the frame shown below, this is:

```
003C2000^SETTEE, YELLOW, OAK^18^DN/1/69^10000^30^1000^9257^_
```

* The four-byte control field, 003C in this example, tells us that the item is 3C (hexadecimal) or 60 (decimal) bytes long. This count is maintained by the system whenever the item is written to disk and records the actual physical length of this item.

* The *end-of-data marker*, or end-of-group marker, indicating that there is no further data belonging to this group in the frame. This is the segment mark.

The physical layout can be seen if we dump a frame which which contains some of the items of our file. For example, the command:

```
DUMP 87654 (X
```

will display the contents of the frame 87654 which the GROUP or ITEM command might have told us holds some of the items of our file. The left section shows the hexadecimal data, and the left section shows the character format.

Typical output might look like this:

```
FID:  87654 : 0 0 0 0 ( 15666 : 0 0 0 0 )

  1 :003C2000^SETTEE, YELLOW, OAK^18^DN/1/69^10000^30^1:
 51 :000^9257^_003D3000^SIDEBOARD, BLUE, ASH^58^MN/5/56:
101 :^13000^15^2000^9272^_00361234^CHAIR, RED, OAK^7^DN:
151 :/5/88^1000^30^2000^9234^_00382222^DESK, BLUE, OAK^:
201 :98^MN/16/92^5600^30^1000^9246^__.................:
251 :................................................:
301 :................................................:
351 :................................................:
401 :................................................:
451 :................................................:
```

The top line shows the FID (and any forward and backward

linkage fields, which are 0 since the data in this frame does not flow from or into any other frames); the data in parentheses is the hexadecimal equivalent of the decimal information.

Access

Access is a very powerful feature of the Pick system which allows you to make enquiries about the data on your files and to produce reports simply by typing in English-language sentences.

Once you have logged on to the Pick system, and the system displays the prompt:

```
:
```

you may type in any Access sentence and this will be processed and the answer - or the report - will be displayed on your screen or printed on the printer.

For example, a sentence such as:

```
LIST STOCK DESCRIPTION PRICE
```

might display a report starting like this:

```
PAGE   1                                  09:29:59  29 JUL 1993

STOCK..... DESCRIPTION.............  PRICE

8888       DESK, GREEN, ASH            56.00
5500       TABLE, YELLOW, PINE         50.00
2000       SETTEE, YELLOW, OAK        100.00
1111       STOOLS, GREEN, PINE         20.00
9000       SIDEBOARD, YELLOW, ASH     130.00
8000       SETTEE, RED, MAPLE          10.00
1000       DESK, GREEN-BLUE, ASH       56.00
```

showing the item-ids of the items on the STOCK file, together with the DESCRIPTION and the PRICE of each item. As we shall see later, words such as DESCRIPTION and PRICE are *datanames* which identify parts of the data on the STOCK file. Your Systems Analyst will tell you which files and which datanames are available to you. These datanames will have been set up for you on a *dictionary* for the file.

Each file has its own dictionary. For example, the dictionary for the STOCK file might hold datanames such as DESCRIPTION, PRICE, QUANTITY, MINIMUM-LEVEL, VALUE, LOCATION, and SUPPLIER, whilst the dictionary for the STAFF file might have datanames such as NAME, SURNAME, INITIALS, ADDRESS, TELEPHONE-NUMBER, START-DATE, DATE-OF-BIRTH, and so on.

If we had typed the sentence:

```
LIST STOCK DESCRIPTION PRICE LPTR
```

adding the word LPTR to the sentence, then the same report would have been produced on the *printer.*

One of the beauties of Access is that it is flexible; you can ask for the datanames in any order, and you can ask for as many datanames as you want. If we had typed the sentence:

```
LIST STOCK PRICE DESCRIPTION QUANTITY
```

we might have produced the report:

```
PAGE   1                                   09:29:59  29 JUL 1993

STOCK.....  PRICE DESCRIPTION..............  QUANTITY

8888         56.00 DESK, GREEN, ASH                64
5500         50.00 TABLE, YELLOW, PINE             15
2000        100.00 SETTEE, YELLOW, OAK             18
1111         20.00 STOOLS, GREEN, PINE             63
9000        130.00 SIDEBOARD, YELLOW, ASH          99
8000         10.00 SETTEE, RED, MAPLE               6
1000         56.00 DESK, GREEN-BLUE, ASH            8
```

Each sentence is processed immediately, so if you have made a mistake, the Access processor will tell you and reject the sentence without any fuss.

Because the way in which you use Access to communicate with the computer, Access is sometimes known as a *query language* or an *enquiry language*. An enquiry language makes it much easier to produce the reports and to answer the questions which you need. Before the advent of enquiry languages, the only solution to these problems was to have a programmer write one or more special report-production programs for each user; this would take many days (or even weeks) and would not be as flexible as the Access enquiry language.

Some versions of the system use other names for the Access language. You may know the language as Access, Recall, RetrieVe, or Info/Access. Although we shall use the name *Access*, most of the material presented here (unless indicated otherwise) applies equally to all versions of the language. If you go on to use enquiry languages on other computer systems such as DOS or Unix, you may encounter other languages such as SQL and dBase, and you will see how much these languages have in common with each other and with Access.

This subject is covered separately in the *MB-Guide to Access sentences* and the *MB-Guide to definitions and dictionaries.*

TCL

The TCL - terminal control language - is made up of a large numbers of verbs. These verbs (or commands which use the verb) are typed in whenever the system is at TCL and waiting for you to give a command. Each verb does something special and a number of options are available to extend the main action of the verb.

We have already met a number of the *commands* - or *verbs* - which you use to control the Pick system.

There is a slight, almost academic difference between a *verb* and a *command* in that the verb is the first - and most important - word, such as:

```
COPY
```

whereas a *command* consists of a *verb* plus some other information, such as:

```
COPY STOCK 1000
```

This set of verbs and commands make up what is known as TCL, the *terminal control language.*

You can issue any TCL command whenever the system displays the prompt:

```
:
```

When the system is waiting for you like this, it is said to be *at TCL* or *at TCL level.*

The TCL language is the fundamental means of using the system. The commands within the TCL language provide a large number of facilities, and we do not have space to discuss them all here. Included amongst the TCL facilities are:

* Creating, maintaining and deleting accounts, files and items;

* Enquiring and producing reports about accounts, files and items;

* Handling the backing storage device;

* Transferring data between the backing storage device and the main memory;

* Setting and enquiring about the status of the system;

* Logging off and logging to other accounts;

* Performing simple arithmetic operations;

* Executing Basic programs to perform specific application requirements.

The MD entries for the TCL verbs are just ordinary Pick

items. You can look at the items on your MD. There is nothing mysterious about their content. If you look at any of the MD items for the verbs, you will see that they all have the letter P as the first letter of attribute 1. Because verbs are simply *items on the MD* you can remove verbs by deleting the item and you may rename verbs by saving the item under another name. Any changes to the verbs on your MD should only be made by the system manager who is responsible for your computer system.

Strictly speaking, there are of five types of TCL command:

1) Access verbs, such as LIST, SORT, COUNT.

2) TCL I verbs - those which *do not* require the name of a file - such as DUMP, POVF, DTX, XTD.

3) TCL II verbs - those which *do* require a file name - such as EDIT, COPY, JET-EDIT.

4) Procs: these provide additional facilities not offered by the standard Pick TCL verbs, such as CT, LISTDICT, LISTFILES.

5) Catalogued programs: these are Basic programs which have been presented in such a way that they can be invoked simply by typing the name of the program. These are almost always specific to an installation.

It is not necessary for you to know the distinction between the various types, and most users are not concerned whether a certain feature is provided by a system routine - such as Access or TCL I and TCL II verbs - a Proc or a catalogued program.

You can use the LISTVERBS command if you want to know what verbs are available to you. The LISTVERBS command shows all the verbs which are available on the account.

The reference literature for your own system will give full details of this topic.

6.1 COPY verb

The COPY verb is particularly valuable because it allows the user to copy items from one file to another.

The general form of the command is:

```
COPY source.file source.item.list (options)
```

where: *source.file* is the name of the file from which the items are to be taken, and *source.item.list* is the list of item-ids of the items which are to be copied. If the source.file is missing, then the process will abandon. If the source.item.list is missing, then the process will ask for the ITEM NAME. If the source.item.list is * then all the items will be taken from the source.file.

Several options are available, of which the most

frequently-encountered are:

D to delete the source items after a successful copy. This cannot be used with the T or P option.

N to suppress the pause at the end-of-page when displaying on the terminal with the T option.

O to overwrite an existing item on the target file with the same item-id. This cannot be used with the T or P option.

P to print the items on the printer. This is how you can print the contents of the items on a file.

T to display the items on the terminal. This is how you can display the contents of the items on a file.

Some examples are:

```
COPY FILE1 *
```
to copy all the items from FILE1 to another file.

```
COPY FILE1 * (T
```
to display all the items from FILE1 on the terminal.

```
COPY FILE1 ITEM1 ITEM2 ITEM3 ITEM4
```
to copy the four specified items from FILE1 to another file.

```
COPY FILE1 ITEM1 ITEM2 ITEM3 ITEM4 (T
```
to display the four specified items from FILE1 on the terminal.

```
COPY FILE1 ITEM1 ITEM2 ITEM3 ITEM4 (P
```
to print the four specified items from FILE1 on the printer.

```
COPY FILE1 ITEM1 ITEM2 ITEM3 ITEM4 (D
```
to copy the four specified items from FILE1 to another file and delete the source items when they have been successfully copied.

A select list may feed the COPY statement:

```
SSELECT FILE1 WITH BALANCE = "0'
COPY FILE (D
```
to copy those items which satisfy the criterion (BALANCE = "0") to another file and delete the source items when they have been successfully copied.

Unless the T and P options are used, the process assumes that the items are to be copied to a target.file and asks for the name of the target.file and the target.item.list of the items:

```
TO:
```

The user must then type in the destination of the items. There are several possible responses:

1) Response 1:

<Return>

in which case the items will be displayed on the terminal (as with the T option).

2) Response 2:

(target.file

For example:

(FILE2

in which case the items will be copied to the target.file FILE2 where they will be retained with their original item-ids. If the O option has been specified, then any existing item-ids on the target.file will be overwritten. Otherwise, a message will be displayed and the item will be ignored.

NOTE THE OPEN BRACKET TO INDICATE THAT A FILE NAME FOLLOWS.

3) Response 3:

(target.file target.item.list

For example:

(FILE2 ITEMA ITEMB ITEMC

in which case the item-ids in the target.item.list - ITEMA, ITEMB, ITEMC - will be used to retain the items on the target.file FILE2.

NOTE THE OPEN BRACKET TO INDICATE THAT A FILE NAME FOLLOWS.

4) Response 4:

target.item.list

For example:

ITEMA ITEMB ITEMC

in which case the item-ids in the target.item.list will be used to retain the items on the source.file.

There should be a one to one correspondence between the item-ids in the source.item.list and those in the target.item.list.

Note that only the open bracket distinguishes the form:

(FILE2 ITEMA ITEMB ITEMC

from:

ITEMA ITEMB ITEMC ITEMD

If the open bracket is omitted, the response will be assumed to be of the second type and the items will be copied back on to the source.file. THIS IS A COMMON SOURCE OF ERROR.

6.2 CT / LIST-ITEM verbs

There are several verbs for printing and displaying the contents of the items on your files. These are used in commands such as:

CT file.name item.id
to display the contents of the item on the Terminal screen.

```
>CT STOCK 1000

    1000
001 DESK, GREEN-BLUE, ASH
002 8
003 MN/17/81
004 5600
005 30
006 2000
007 9278
```

CT file.name item.id1 item.id2
to display the contents of several items on the screen.

CT file.name *
to display the contents of all the items which are held on the file.

CT file.name item.id (P
to print the contents of the item on the printer.

CT file.name * (P
to print the contents of all the items which are held on the file.

CT DICT STOCK DESCRIPTION
to display the contents of the DESCRIPTION dictionary definition item for the STOCK file.

LIST-ITEM file.name 'item.id'
to display the contents of the item on the screen.

LIST-ITEM file.name
to display the contents of all the items which are held on the file.

LIST-ITEM file.name (P
to print the contents of all the items which are held on the file.

SORT-ITEM file.name (P
to print the contents of all the items which are held on the file. The report will be sorted into order of the item-ids.

6.3 TCL options

There are a number of options which can be applied to the TCL commands. The range and action of most options vary according to the particular verb, whilst some options are available with most verbs. For example:

```
(P
```

may be used in a command such as:

```
LIST STOCK (P
```

and will direct the output from this and almost any command to the printer, instead of displaying the output on the screen, and:

```
(N
```

as in:

```
LIST STOCK (N
```

will suppress the normal end-of-page pause when you are displaying output on the screen.

These and many other options may be added to the TCL commands which you will use. You will become familiar with the options which are associated with the TCL commands which you will use. If you are uncertain, the reference manuals for your system will indicate the options which are available with each command and what effects those options have.

6.4 Editor

The Pick editor is a line editor and will allow the user to edit any item on any file. It is particularly useful for:

* Creating and maintaining Basic programs and Procs.

* Creating and maintaining Access attribute-definitions on the DICT of your data files.

* Creating, changing and deleting test data items. Real data items will always be maintained by the programs within your application system.

The Editor is invoked by TCL commands such as these:

```
EDIT MYPROGS PROG01

EDIT MYFILE ITEM1 ITEM2 ITEM6

EDIT MYPROGS *

SSELECT MYPROGS = "[AMEND]"
EDIT MYPROGS
```

If you omit the item list and no select list is available,

then the editor will ask you for:

```
ITEM ID:
```

If you invoke the editor with a list of items to be processed, as in the first two examples above, then, when the editing of an item is terminated (with an FI or FD command), the next item in the list will be submitted for editing until the list is exhausted.

We discuss the Editor in more detail in the *MB-Guide to Using the Editor*, and in your reference literature.

6.5 Notes about the Pick editor

An editor command may be entered in either lower case or UPPER CASE. So that:

```
.DE999
```

and:

```
.de999
```

have the same effect.

The displayed output from an editor command, such as:

```
.R999/abc/def/
```

may be suppressed by putting a full stop before the command.

```
.R999/abc/def/
```

or:

```
.r999/abc/def/
```

in which case the lines will not be displayed as they are changed.

6.6 EDIT commands

In this and the following sections, we present a summary of the Pick editor commands which you will encounter. We look first at the more frequently used EDIT commands. We show the dot prompt with these commands, although the prompt is displayed by the Editor and is not entered as a part of the command.

.n

move the pointer to line number n. this is identical to the Gn command.

.<Return>

advance the pointer a line at a time.

During a sequence of data lines following an I or R command, a <Return> indicates the end of the data lines.

.A

repeat the last L command again.

.B

move the pointer to the bottom of the item, the EOF or EOI.

.DE{n}/sssss/{p{-q}}
delete all those of the next n lines, including the current line, which contain the string sssss in columns p to q inclusive. For example:

.DE5/HELP/
will delete every one of the next five lines which contains the character string HELP. The final delimiter may be omitted:

.DE5/HELP
unless a range of columns is specified:

.DE7/HELP/10-99
will delete every one of the next seven lines which contains the character string HELP 2 anywhere in columns 10 to column 99 inclusive.

If you omit the number of lines:

```
.DE/PRINT/
```

then the action differs according to the implementation.

On some, this will delete the current line if it contains the specified string, the command is thus interpreted as:

```
.DE1/PRINT/
```

On others, the editor will scan forwards from the current line, deleting the first line which is found to contain the string.

The following special forms of this command are frequently used:

.DE
Delete the current line.

.DEn
Delete n lines starting with the current line. For example:

.DE7
will delete the current line and the next 6 lines.

.DEn/
Display and delete n lines starting with the current line. For example:

.DE7/
will display and delete the current line and the next 6 lines.

.EX
Exit from the editor and abandon the processing of the current item.

.EXK

Exit from the editor and abandon the processing of any further items in the list item ids.

.F

Move the pointer back to the top of the item and incorporate any amendments made, and then continue with the editing of the item.

.FD

Delete the current item from disk and terminate the editing of the item.

If an item has been accidentally deleted by means of the editor FD command, then it may be recovered by using the RECOVER-FD verb, as discussed later.

.FI

File the item, implement any changes made to the item, write the item back to the disk, overwriting the original version if there is one. If you are editing a list of items, then the next item will be submitted to the editor for processing, otherwise the editor processing will terminate.

.FS

Implement any changes made to the item, write the item back to the disk, overwriting the original version if there is one, and continue with the editing process, unlike FI which exits from the editor .

There are many extensions to these FD / FI / FS commands, but we do not have space to discuss them in detail here.

.Gn

Move the pointer to line number n.

.I

Prepare to insert one or more lines after the current line.

The editor will display a prompt of the form:

```
00n+
```

to show that lines are being inserted after line number n.

You can then enter the new information, line by line, each line being prompted by the same string:

```
00n+
```

An input of null will terminate the insert action.

Care must be taken when it is required to insert blank (null) lines. The situation may be solved by inserting lines containing a dummy character string which is then replaced by null, as illustrated here where we wish to insert three blank (null) lines after line number 5:

```
.G5
.I
005+ZZZ
005+ZZZ
005+ZZZ
005+                                        <Return> only
.F
.G6
.R3/ZZZ//
006
007
008
.
```

.I xxxxx
Insert the single line of data xxxxx after the current line. There is a space between the I and the data. For example:

.I PRINT "ENTER NEW ADDRESS"
will insert the line:

```
PRINT "ENTER NEW ADDRESS"
```

immediately after the current line.

.Ln
Display n lines starting at the line following the current line. For example:

.L20
will display the 20 lines following the current line. If you omit the number of lines, then the editor will display the next line only.

.L{n}/sssss/{p{-q}}
displays all those of the next n lines which contain the string sssss in columns p to q inclusive. For example:

.L20/AREA/
will display those of the next 20 lines which contain the string AREA in any position in the line.

.L5/AREA/6-12
will display those of the next 5 lines which contain the string AREA within columns 6 to 12 of the line.

.L/AREA/
will scan the item and display the first line which was found to contain the string AREA in any position in the line.

.L:xxxxx
Scan the item and display the first line found to begin with the characters xxxxx.

6.7 ME command

The ME command will import (or merge) text from another item into the current item. This is particularly useful for

copying text which you have prepared elsewhere.

.ME{n}/iiiii/{m}
will merge n lines from item iiiii (which is to be found on the same file as the current item), starting at the line number m in the source item. The text is inserted immediately after the current line in the item being edited. Here are some examples.

.ME5/ANDATA/10
will copy the 5 lines from the item ANDATA, starting with line 10 (that is, lines 10, 11, 12, 13, and 14) and insert them after the current line in the item which you are editing. The item ANDATA will be unchanged. If you omit the item id, like this:

.ME5//10
then this will copy lines 10, 11, 12, 13 and 14 from the current copy of the item which you are editing and insert them after the current line.

If the item is to be taken from another file, then the form:

.ME{n}(fffff {iiiii}){m}
will merge n lines from item iiiii on the file fffff, starting at line number m, and insert them after the current line in the item being edited. For example, the command:

.ME5(ALLPROGS UPDP)10
will copy lines 10, 11, 12, 13 and 14 from the item with item id UPDP on the file ALLPROGS.

6.8 R command

The R command is one of the most valuable Editor commands. Its purpose is to change the text of the line(s) of the current item, replacing one string of characters by another.

.R{n}
allows you to type in n lines of information which are to replace the next n lines of the item, starting with the current line.

.R{n}/string1/string2/{p{-q}}
replaces the string string1 by the string string2 for the first (left most) occurrence in columns n lines starting at the current line.

Note the following points about this form of the R command:

1) If you omit the number of lines, then a value of 1 is assumed. This is standard for the editor.

2) The strings may be of different lengths.

3) Either string may be null.

4) In the examples shown here, we use the / character to separate the strings, although any character (other than the colon and numeric digits) may be used. The

separator character must not appear within either string.

5) If the character ^ is used within string1, it has the effect of a *wild-card* character. However, the ^ command described below may be used to override this wild card effect.

6) An attribute mark, that is, <Ctrl> and ^ character, within string2 will terminate the line at that point.

Here are some examples:

.R/WAGE/SALARY/
will replace the first occurrence of the character string WAGE by the string SALARY in the current line.

.R5/WAGE/SALARY/
will replace the first occurrence of the string WAGE by the string SALARY in the next five lines, starting with the current line.

.R/WAGE//
will remove the first occurrence of the string WAGE from the current line.

.R/ /WAGE/
will append the string WAGE to the right hand end of the current line. The string of spaces must be longer than any string of spaces which occurs within the line. The AP command may also be used, as described below.

.R//WAGES/
will insert the string WAGE at the left hand end of the line.

.R/WAGE/SALARY/30-50
will replace the first occurrence of the string WAGE in columns 30 to 50 (inclusive) by the string SALARY in the current line.

.R5/WAGE/SALARY/30-50
will replace the first occurrence of the string WAGE in columns 30 to 50 (inclusive) by the string SALARY in the next five lines, starting with the current line.

.R/WAGE/SALARY/30
will replace the first occurrence of the string WAGE in columns 30 onwards by the string SALARY in the current line.

.R/*/-/30-30
will replace any * in column 30 by the - character. Any * elsewhere in the line will be ignored.

.R5/WAGE/SALARY/30
will replace the first occurrence of the string WAGE in columns 30 onwards by the string SALARY in the next five lines, starting with the current line.

.R999/^//30
will delete the 30th character from this and the next 998

lines. The ^ character here is the normal ^ above the figure 6 on the keyboard.

.R999/STOP/^/
will delete all the text beyond the word STOP in the current line. The ^ character here is created by typing the sequence:

<Ctrl> ^

where ^ is the normal character.

.RU5/WAGE/SALARY/
either of these will replace all occurrences of the string WAGE by the string SALARY in the next five lines, starting with the current line.

.RU5/WAGE/SALARY/20-90
either of these will replace all occurrences of the string WAGE by the string SALARY in columns 20 to 90 of the next five lines.

6.9 Other commands

.T
move the pointer back to the top of the item (line 000). any amendments which have been made will not be seen until an F or FS command is issued.

.X
cancel the effect of the last ME or R or I command.

.XF
is available on some implementations and will allow you to cancel the effect of all changes made since the last F or FS command.

6.10 EDIT commands : miscellaneous commands

in this section, we describe the remaining commands. You may find them of use once you have mastered the elementary set described earlier.

The editor prestore commands are discussed later.

.?
display the current line number. On some implementations, the file name and item id are also displayed. This is an alternative to ?I below.

.?I
display the file name and the item id of the item which is being edited. On some implementations, the size of the item and the current line number are also displayed. This is an alternative to ? above.

.C
display the column mask showing the column numbers. This is useful when replacing a specific string in a specific position.

.Nn
advance the pointer by n lines.

.S?
display a message showing the current size of the item being edited, and the amount of workspace available for expansion of the item. This is an alternative to ?S on some implementations.

.S
switch the line number display off and on.

.TB a b c ... Z
set tabulator stops at columns a, b, c and so on, allowing the <Tab> key to be used to shift input text along to the next available tab setting. This is useful when entering Basic programs or tables of information to be processed by Runoff.

.Un
move the pointer back n lines up the item. The command U, without a numeric parameter, has no effect.

.Z
reset the zoning set by a Zp-q command and is equivalent to Z1-999.

.Zp-q
display only columns p to q of the lines of the item. For example:

.Z1-70
will truncate the displayed lines at column 70, although the entire line will be considered when issuing commands such as L and R.

.^
set and unset a switch which indicates that the character ^ is to be regarded as a normal keyboard character and not as the attribute mark.

6.11 EDIT prestore commands

If you are using a complicated editor command over and over again, then you may store the command and recall it when required.

Any editor command can be stored by entering a P, followed by the command to be stored. For example:

```
.P L23
```

will store the command L23. Some implementations need a space before the command, others do not.

This stored command may then be activated by giving the command P alone:

```
.P
```

and this will then have the same effect as issuing the original command:

```
.L23
```

Up to ten stored commands may be active at any one time, each being identified by its store number: P, P1 through to P9. P0 and P are equivalent. The terms *prestore* and *prestored commands* are used in this context.

Thus, we might prestore a command such as:

```
.P6 R999/!/./
```

and then invoke this, as and when required, by:

```
.P6
```

Most implementations usually apply the store command:

```
.P L23
```

on entry to the editor, so that the simple:

```
.P
```

command will display up to one screen full of data. The contents of the other prestores will be as left when they were last used on the current port. The PD command can be used to check the contents.

All ten prestores are used in the same way, and each will generally be used to store a separate command. For example, let us imagine that we wish to change the string WAGE to the string SALARY in attribute 7 of an item, then we might store the commands:

```
.P1 G7
.P2 R/WAGES/SALARY/
```

and then every time you issue the command:

```
.P1
```

this will move the pointer to line (attribute) 7 of the item, and:

```
.P2
```

this will change the string WAGE to the string SALARY.

It is possible to store a sequence of several commands in one prestore. This is done by separating the individual commands by the <Esc> escape character:

```
[
```

The <Esc> character is entered by the ESCAPE key or by the sequence <Ctrl>[on some keyboards.

An example of such a command might be:

```
.P1 G7[R/WAGES/SALARY/
```

You may even *program* the Pick editor so that a particular sequence recalls itself. For example:

```
.P1 R999/WAGES/SALARY/[FI[P1
```

This is particularly useful when making a number of changes to a list of items, a process which otherwise might need a complex program to scan and change the items. This is illustrated by the following sequence:

```
:EDIT PAYROLL *
.P1 G3[R/PERSONNEL/P[F
.P2 G3[R/ADMINISTRATION/A[F
.P P1[P2[FI[P3
.P
```

The effect of this sequence is change the string PERSONNEL to P, the string ADMINISTRATION to A in attribute 3 of all items on the PAYROLL file.

6.12 RECOVER-FD

If an item has been accidentally deleted by means of the editor FD command, it may be recovered by using the RECOVER-FD verb immediately after the editor has terminated. The RECOVER-FD verb requests:

```
INPUT ITEM ID*
```

to which you must enter the item id of the item which was deleted. The item will be then be saved on the original file. The state of the recovered item will be as prior to the last use of the editor.

RECOVER-FD is a TCL verb and the recovery must be done immediately after the deletion. If any other activity is performed before the RECOVER-FD is issued, then the item may be irretrievable.

6.13 TCL stacker

The TCL stacker - sometimes called the *dot processor* - is a feature which allows you to recall the last TCL command which you entered and issue it again. A TCL stacker is available on many implementations of the Pick system.

The actual commands which you use to recall and amend your commands will differ, depending upon which system you are using. Details of what *your* TCL stacker can do will be displayed if you enter the command:

```
.?
```

Those presented here are typical.

The TCL command:

```
.1
```

will recall the last TCL command which you entered; the command:

```
.2
```

will recall the next-to-last TCL command which you entered, and so on. Elsewhere, you might encounter the commands:

```
.X1
```

and:

```
.X2
```

for the same purpose. In some cases, the command will be *displayed* and then you may amend the command by means of the <Backspace> key or by typing extra text on the end.

If you want to see what commands are stored, then the command:

```
.L
```

will display the current contents of the stack. Up to 20 TCL commands will be held at any one time. A command such as:

```
.L9
```

will display the contents of entry number 9 in the stack.

If you want to edit the contents of the stack, then the command:

```
.E
```

will pass the current contents of the stack for you to amend by means of the Editor.

The TCL stacker may also have facilities for amending the commands in the stack and for *saving* specific commands (or a sequence of commands) permanently on a file, and there may also be facilities for recalling such saved commands later.

Although its general principles are the same, the contents of the TCL stacker on Advanced Pick are handled differently from those outlined here.

Basic

Basic is a version of the original Dartmouth Basic which has been extended to allow the programmer to take advantage of the features of the Pick system. There are major differences between Basic and the dialects which are available in other environments such as DOS.

Each Basic program is held as a separate item on a file, and comprises a sequence of Basic statements, the *source program.* The source program is created and maintained by one of the standard editors and converted into an executable form, the *object program,* by the compiler. This object program can then be executed, whenever required.

The language is covered in more detail in a number of *MB-Guides* including the *MB-Guide to the Basic language.* Here, we shall look at a few general points about the Basic language.

7.1 Looking at Basic

Diagram 1 shows a typical Basic program - which I've called CALC01.

```
    PROGFILE
000 CALC01
001 * Program to demonstrate some features
002 * of the Basic programming language
003 *
004 * Version A
005 ******************************************************
006 10   PRINT 'What is your name':
007      INPUT NAME
008      IF NAME='END' THEN STOP
009      PRINT 'Hello, ':NAME:'!'
010 * Ask for the range of numbers
011 22   PRINT 'What is the end of the range of numbers':
012      INPUT FINISH
013      IF FINISH='END' THEN STOP
014      IF NUM(FINISH) THEN GOTO 30
015      PRINT 'You must enter a number'
016      GOTO 22
017 30 * Clear the screen
018      PRINT @(-1):
019 * Calculate and display the numbers
020      PRINT 'Square root','Number','Square','Cube'
021      PRINT
022      FOR X=1 TO FINISH
023          PRINT SQRT(X), X, X*X, X*X*X
024      NEXT X
025 33   PRINT
026 * Another range of numbers?
027      PRINT 'Do you want more numbers, ':NAME:
028      INPUT RESPONSE
029      IF RESPONSE='YES' THEN GOTO 22
030      IF RESPONSE='END' THEN GOTO 99
031      IF RESPONSE NE 'NO' THEN GOTO 33
032 99   PRINT
```

```
033       PRINT 'Thank you, ':NAME
034       END
```
Diagram 1: A Basic program

This program has been established as an item on a file called PROGFILE where it has been saved with the item-id CALC01. When we execute this program, the screen display might look like that in Diagram 2.

```
What is your name?Bill
Hello, Bill!
What is the end of the range of numbers?14

Square root        Number              Square              Cube

1                  1                   1                   1
1.4142             2                   4                   8
1.732              3                   9                   27
2                  4                   16                  64
2.236              5                   25                  125
2.4494             6                   36                  216
2.6457             7                   49                  343
2.8284             8                   64                  512
3                  9                   81                  729
3.1622             10                  100                 1000
3.3166             11                  121                 1331
3.4641             12                  144                 1728
3.6055             13                  169                 2197
3.7416             14                  196                 2744

Do you want more numbers, Bill?NO
Thank you, Bill
```
Diagram 2: The program output

Many of the statements are self-explanatory, but the following points are of interest:

* The numbers shown down the left of this list (001 to 034) are the *line-numbers* of the individual lines in the program, shown only when we display the source program. These numbers are not a part of the text of the program itself.

* The lines which begin with an asterisk are *comments* and we include these to explain what the program is doing. The computer ignores these comments when it is executing the program.

* Some lines begin with a number; these are *statement labels* and they serve as destinations when we want to jump about the program with GOTO and other statements. Here we use statement labels:

```
10
22
30
33
99
```

Many programmers have reservations about the use of GOTO statements. We address this in a moment, when we look at an alternative version of the program.

* A PRINT statement such as:

```
PRINT 'What is your name':
```

displays the message:

```
What is your name
```

on the screen. The final colon holds the cursor in position after the word *name*.

* The INPUT statement:

```
INPUT NAME
```

displays a question mark and waits for the user to type in some information, and when he/she presses the <Return> key, that information will be placed in a *variable* called NAME.

* The IF statement:

```
IF NAME='END' THEN STOP
```

tests whether the contents of the variable called NAME (just typed in by the user) is the word:

```
END
```

and, if so, the STOP statement is executed and the program terminates.

* The statement:

```
PRINT 'Hello, ':NAME:'!'
```

again displays something on the screen. This time, it displays a message made up of the word:

```
Hello,
```

followed by the contents of the variable called NAME, followed by the exclamation mark; the colon is used to join the parts together. So, when our user entered his name as:

```
Bill
```

this PRINT statement displayed:

```
Hello, Bill!
```

* The statement:

```
IF NUM(FINISH) THEN GOTO 30
```

can be read as "if the information which is stored in the variable called FINISH is a *number*, then go to the statement labelled 30, otherwise carry on to the following statement".

* The statement:

```
GOTO 22
```

causes the processing to jump to the statement labelled 22 (at line number 011). Note that this refers to the statement label and not the sequential line-number 022. Statement labels may be numeric, as here, or they may be alphanumeric, as in the program in Diagram 3 below.

* The statement:

```
PRINT @(-1):
```

is a special usage of the PRINT statement which clears the screen.

* The statement:

```
PRINT 'Square root','Number','Square','Cube'
```

is another form of the PRINT statement which displays the heading line.

* The next group of statements:

```
FOR X=1 TO FINISH
    PRINT SQRT(X), X, X*X, X*X*X
NEXT X
```

are particularly interesting because they cause the statement at line 23 to be repeated for the number of times requested by the user.

* The FOR statement and the NEXT statement will always appear as a pair like this, controlling the repetition of the statements in between.

* If our user had answered 32 when the program asked:

```
What is the end of the range of numbers
```

then the number 32 would be stored in the variable called FINISH, and the FOR and NEXT statements would cause the statement:

```
PRINT SQRT(X), X, X*X, X*X*X
```

to be repeated, with the variable called X taking all the values from 1 to 32.

* In processing the PRINT statement, the program will calculate the three values SQRT(X) (that is, the square root of the number held in X), X times X (that is, X squared), and X cubed, and then display these on one

line.

* At the moment when the variable called X contained the number 12, the displayed line would show the four values 3.4641, 12, 144, 1728.

* The final statement:

```
END
```

indicates that this is the end of the program and, when it is executed it causes the program execution to stop (exactly like the STOP statement which we saw earlier).

* We can put spaces within the statement; such *indentation* makes it easier to read.

* The text of the statements is normally entered in upper-case letters.

Although this program works perfectly well, it would be much easier to understand and to amend if it were written as a structured program. This is possible with Basic, as this alternative version of the program illustrates. Program CALC02 in Diagram 3 shows the same processing written as a series of *subroutines* and using some of the *program structures* which Basic offers. The use of subroutines greatly clarifies the action of the program and makes it easier to read and understand; each separate subroutine is a self-contained piece of coding which goes about its business without affecting anything else; only the variables NAME and FINISH are used to pass data between the main processing loop (in lines 006 to 018) and the subroutines (in lines 021 onwards). If we ever need to change our program, we can happily amend the coding in the subroutines knowing that we are not likely to affect other parts of the program, provided that the data going to and from the subroutines is as before. It would be less easy to amend the program CALC01.

To reach a subroutine, we use a statement such as:

```
GOSUB GET.USER.NAME
```

and this passes the flow of our processing down to the statement labelled GET.USER.NAME. When we next encounter the:

```
RETURN
```

statement, the flow of processing is transferred back to where it came from and continues from the statement following the GOSUB GET.USER.NAME statement.

In this program, we make great use of a structure known as a *loop*. This has the general form:

```
LOOP
     action A
UNTIL condition is true DO
     action B
```

```
    REPEAT
```

The general effect of the LOOP structure is:

* Carry out action A,

* Test the condition,

* If the condition is *true*, then leave the LOOP and carry on with the statement following the REPEAT;

* If the condition is not *true*, then carry out action B and then go back to the beginning of the LOOP once more.

Look at the coding starting at the statement labelled GET.USER.NAME and follow the action of the LOOP there.

Some of the conditions in our program are fairly complicated, for example:

```
    UNTIL (NUM(FINISH) AND FINISH>1) OR FINISH='END' DO
```

will repeat the LOOP until the data entered by the user (and stored in the variable called FINISH) is either:

* A number *and* greater than 1 (this is a new condition that I've imposed, so that the user doesn't enter 1 or a number less then 1),

or:

* The word END.

You will notice that, unlike CALC01, we have very few GOTO statements in this program. This makes it easier to follow the flow of the processing, and enables us to discover how we reached a certain point in the processing (we might need this information if something has gone wrong with the program and it is not doing the right thing). There is, in fact, only one GOTO statement in this program - at line 013; this handles what is known as an *exception condition* (when the user has asked to abandon the processing) and allows us to leave the main process and go to the end of the job.

```
    PROGFILE
000 CALC02
001 * Program to demonstrate some features
002 * of the Basic programming language
003 *
004 * Version 2 - using subroutines
005 ******************************************************
006       GOSUB GET.USER.NAME
007       GOSUB WELCOME
008       LOOP
009           GOSUB GET.RANGE
010       UNTIL FINISH='END' DO
011           GOSUB CALCULATE
012           GOSUB ASK.FOR.MORE
013           IF RESPONSE='NO' THEN GOTO FINISH
```

```
014       REPEAT
015 FINISH: * End of job
016       PRINT
017       PRINT 'Thank you, ':NAME
018       STOP
019 ****************************************************
020 * Subroutines
021 GET.USER.NAME: ** Get user's name ********************
022       LOOP
023           PRINT 'What is your name':
024           INPUT NAME; NAME=TRIM(NAME)
025       UNTIL NAME NE '' DO
026           PRINT 'You must enter your name'
027       REPEAT
028       RETURN
029 WELCOME: ** Print welcome message *********************
030       PRINT 'Hello, ':NAME:'!'
031       RETURN
032 GET.RANGE: ** Find range of numbers ******************
033       LOOP
034           PRINT 'What is the end of the range of numbers':
035           INPUT FINISH
036       UNTIL (NUM(FINISH) AND FINISH>1) OR FINISH='END' DO
037           PRINT 'You must enter a number'
038       REPEAT
039       RETURN
040 CALCULATE: ** Calculate and display the range ********
041       PRINT @(-1):'Square root','Number','Square','Cube'
042       PRINT
043       FOR X=1 TO FINISH
044           PRINT SQRT(X), X, X*X, X*X*X
045       NEXT X
046       RETURN
047 ASK.FOR.MORE: ** Ask if there are more numbers *******
048       LOOP
049           PRINT
050           PRINT 'Do you want more numbers, ':NAME:
051           INPUT RESPONSE; RESPONSE=TRIM(RESPONSE)
052       UNTIL RESPONSE='YES' OR RESPONSE='NO' DO
053           PRINT 'Enter YES or NO'
054       REPEAT
055       RETURN
056       END
```

Diagram 3: Structured program using in-line subroutines

7.2 Using a Basic program

The development of a Basic program goes through several stages, and the programmer will use a standard set of Pick tools. Each program must be:

1) Written and saved as an item on a file. The Editor is used for this stage:

```
EDIT PROGFILE CALC01
```

2) Compiled:

```
BASIC PROGFILE CALC01
```

3) Executed:

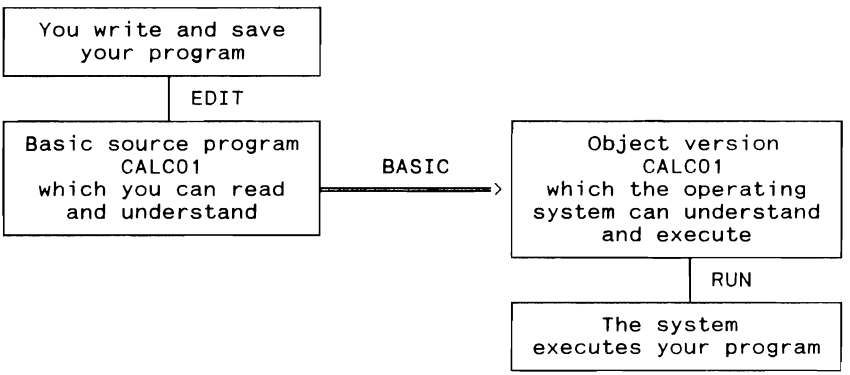
```
RUN PROGFILE CALC01
```

Whenever you make any changes - whether it be to correct a mistake in your calculations, or to add new features to the program - you must go back to stage (1) and change the *source version* of your program.

* Each Basic program is created and saved as an *item* on a program file. This item is your *source program* and consists of statements in the Basic language. We have already seen the source code in item CALC01 on the file PROGFILE.

* The source program must be converted into a form which can be executed. This translation process is known as *compilation* and it performed by a piece of software known as the *compiler*. This executable version of your program is produced by the compiler and is called the *object program*. This will be created as item CALC01 on the DICT section of the file PROGFILE, in our example. If you have made any mistakes, such as spelling the word PRINT incorrectly, then the compiler will reject these and no object program will be produced.

At this stage, there will be two versions of your program: the source version, which you can read and understand, and the object version, which the system can understand and execute.



If a Basic program uses *external* subroutines, such as are called by statements of the form:

```
CALL CALC.SUB.01
```

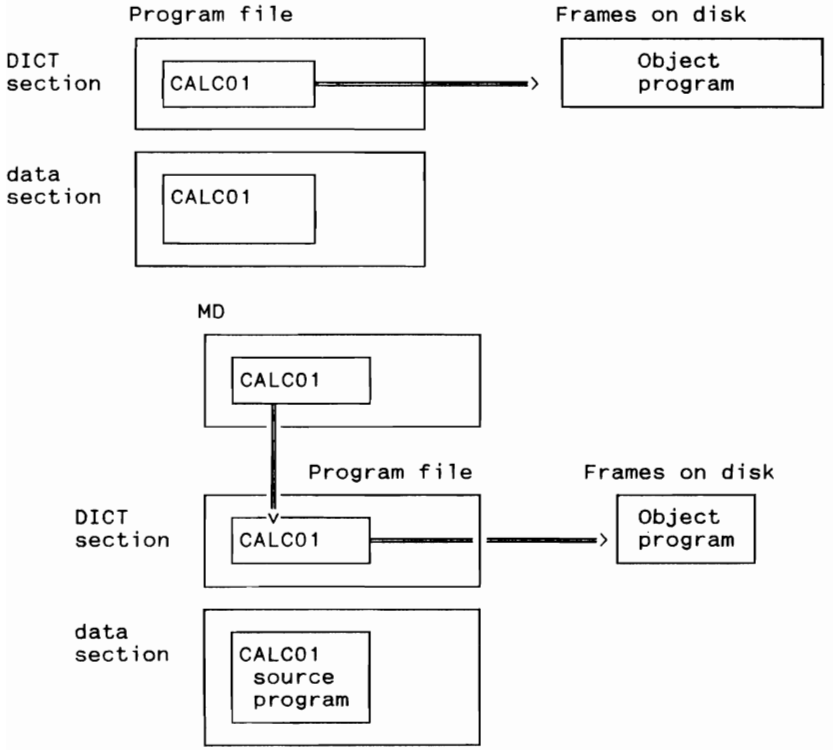
or:

```
CALL CALC.SUB.02(VALUE,RATE,2)
```

then the external subroutine(s) must be *catalogued*. This is achieved by commands such as:

```
CATALOG PROGFILE CALC.SUB.01
```

and places a pointer to the object code on the MD so that the subroutine can be located when required.



7.3 Basic statements

The Basic language offers the following statements:

!	EQUATE	MAT	READV
*	EXECUTE	MATREAD	READVU
ABORT	EXIT	MATREADU	RELEASE
BEGIN CASE	FILE	MATWRITE	REM
BREAK	FOLD	MATWRITEU	RETURN
CALL	FOOTING	NULL	REWIND
CAPTURING	FOR / NEXT	ON ... GOSUB	ROOT
CASE	GO / GOTO	ON ... GOTO	RQM
CASING	GOSUB	OPEN	SELECT
CHAIN	HEADING	OUT	SLEEP
CLEAR	IF	PAGE	STOP
CLEARFILE	IN	PRECISION	SUBROUTINE
COMMON	INCLUDE	PRINT	TCLREAD
COMMON /com/	INPUT	PRINTER	UNLOCK
CRT	INPUT @	PROCREAD	WEOF
DATA	INPUTERR	PROCWRITE	WRITE
DEBUG	INPUTERROR	PROGRAM	WRITET
DELETE	INPUTNULL	PROMPT	WRITEU
DIMENSION	INPUTTRAP	READ	WRITEV
ECHO	LOCATE	READNEXT	WRITEVU
END	LOCK	READT	
ENTER	LOOP / REPEAT	READU	

The reference literature for your own system will give full details of this topic.

7.4 Basic functions

The Basic language offers a great many standard functions for performing specific processing. These are called and used as required, and do not have be included or declared as with some languages.

@	COUNT	ICONV	OCONV	SPACE
ABS	DATE	INDEX	PWR	SQRT
ACCESS	DCOUNT	INSERT	REM	STR
ALPHA	DELETE	INT	REPLACE	SYSTEM
ASCII	DTX	LEN	RND	TAN
CHAR	EBCDIC	LN	SEQ	TIME
COL1	EXP	MOD	SIN	TIMEDATE
COL2	EXTRACT	NOT	SORT	TRIM
COS	FIELD	NUM	SOUNDEX	XTD

The reference literature for your own system will give full details of this topic.

8 Procs

The original purpose of the Proc language was to provide a means of writing a processing sequence which could be used to invoke one or more TCL commands, and, together with the now obsolete *batch processor* which allowed users to add, change and delete items, it served all the fundamental needs of the technical user in the very first implementations of the Pick system. Just as a processing sequence written in the Basic language is called a *program,* so a processing sequence written in the Proc language is called a *Proc.* A great many of the fundamental utilities, such as:

```
LISTFILES
LISTDICT file.name
FILE-SAVE
```

are implemented as Procs. Since its inception, the Proc language has been expanded to encompass many programming features. As a programming medium, however, it has many limitations and it has largely been superseded by the Basic language. Full details can be found in the *MB-Guide to Creating and using Procs* and in your reference literature.

8.1 Proc versus Basic

Since Basic is a very powerful programming tool, why is there a need to persist with Procs? There is still a requirement for the technical user to be familiar with Procs for several reasons, and in certain environments, there is a strong case to be argued for the use of simple Procs by the *non-technical* user.

Let us imagine that a user has a frequent need to dump a copy of the STOCK file to backing storage. The TCL commands to carry out this action are shown in sequence (a) of the diagram below. To automate the process, it is much quicker and simpler to create a Proc, such as that in sequence (b), rather than create the equivalent Basic program, as shown in (c), since the program must be compiled before use and would occupy much more space (for both the source and the object forms of the program) than would the Proc.

a

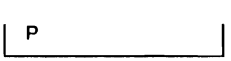
```
TCL commands
>T-ATT
>T-REW
>T-DUMP STOCK
>T-REW
>T-CHK
>T-REW
>T-DET
```

b

```
A Proc
PQN
HT-ATT
P
HT-REW
P
HT-DUMP STOCK
P
HT-REW
P
HT-CHK
P
HT-REW
P
HT-DET
```

c

```
A Basic program
EXECUTE 'T-ATT'
EXECUTE 'T-REW'
EXECUTE 'T-DUMP STOCK'
EXECUTE 'T-REW'
EXECUTE 'T-CHK'
EXECUTE 'T-REW'
EXECUTE 'T-DET'
END
```



The use of a Proc will save the end-user a lot a typing and avoid the chance of making mistakes when the Proc becomes a part of the live application system.

The main *advantages* of the Proc language over Basic are:

+ A Proc is a quick and simple means of issuing (and reissuing) a sequence of one or more TCL commands and/or Access sentences.

+ A Proc is fairly simple to create, amend and use. Non-technical users may need a little practice with the editor and with the concept of Procs before they get it right.

+ The Proc processor is interpretive and the Proc does not have to be compiled each time it is changed and may be used straightaway.

The main *disadvantages* of the Proc language, as compared to Basic are:

- Because the Proc processor is interpretive, any errors will only be revealed as an attempt is made to execute an offending statement.

- It is not easy to read and comprehend a Proc. Even with copious comments, the action of the statements can seem obscure or arcane.

- Since there are no structured programming facilities and all processing control is achieved by means of GOTO and subroutine call statements, maintenance can be difficult, not to say dangerous.

- Only simple arithmetic can be performed, adding / subtracting a constant from a value held in the input buffer.

- Proc has few data and string handling capabilities.

- The concept of program variables is replaced by a set of dynamic-array *buffers* in which each element is identified by its position. This is prone to errors and mistakes.

All these points - especially the last one - make it very difficult (and daunting) for anyone attempting to modify someone else's (or even their own) Procs.

8.2 The structure of a Proc

A Proc is an item consisting of a sequence of *Proc statements*. The statements are written one statement to one attribute (or line) of the item.

Let us look at a simple Proc which will issue the Access

sentence which we would type in as:

```
SORT STOCK WITH LEVEL = "0" DESCRIPTION LEVEL
```

We might be in a situation where one of our users, Sally, needs to produce this report every day and, in order to make her life simpler, she has asked us to set up a Proc which will allow her to type in:

```
SALLY001
```

and the report will be produced. The Proc might look like this:

```
    MD
000 SALLY001
001 PQ
002 H SORT STOCK WITH LEVEL = "0" DESCR LEVEL
003 P
```

There are three important elements in this Proc:

* The PQ statement tells the system that this is a Proc;

* The Proc statements beginning with the letter H build up a string of characters in a storage area known as the *output buffer*. Each H statement appends another string of characters to the end of the output buffer. We could build up the output buffer by issuing one long command (as in the first example), or by using a sequence of shorter strings.

* Finally, the P statement processes the command which has been constructed in the output buffer.

We would create a Proc such as this and save it on the MD, with the item-id SALLY001 in this case. Whenever she needs to produce the report, Sally will type in the command:

```
SALLY001
```

and the Access report will be produced immediately.

A Proc may issue several TCL commands and/or Access sentences, like this:

```
    MD
000 SALLY002
001 PQ
002 H SORT STOCK WITH LEVEL = "0" DESCR LEVEL LPTR
003 P
002 H SORT STOCK WITH LEVEL < "0" DESCR LEVEL LPTR
003 P
002 H SORT STOCK WITH LEVEL > "0" DESCR LEVEL LPTR
003 P
002 H OFF
003 P
```

Procs can - and do - become more complicated, offering facilities for:

* Displaying and printing messages and results.

* Carrying out tests and decisions on the data.

* Picking up data from the TCL command which invoked the Proc.

* Asking the user for data as the Proc executes.

Indeed, Procs are able to perform many of the actions of a Basic program ... but, as we have said, a Basic program is much easier to read, write and maintain than would be the equivalent Proc.

8.3 Proc statements

The Proc language offers the following statements.

```
+n              H{x}<<          O               ST OFF / ON
-n              H{x}<           P               STOFF
A{s}{n}{,m}     Hx              PH              STON
B               IF A{n}{,m}     PP              T
BO              IF {£} A{n}     PQ              Uxxxx
C               IF {£} E        PW              X{x}
D{n}{,m}{+}     IF {£} S        PX              (f i)
F               IHx             RI{n}           [f i]
G n             IN{p}           RO              []
GO n            IP{p}           Sn
GO A{n}         IS{p}           SP
GOTO n          IT              SS
```

The reference literature for your own system will give full details of this topic.

9 Runoff

The Runoff text processing system is a standard part of the Pick system, and is provided for the production of documents, manuals, specifications, and any similar printed matter.

It can also be used as a tool within Basic programs, displaying and/or printing messages, explanations, help texts and so on. Using Runoff in this way reduces the programming effort in formatting and outputting the text, and the fact that the text is held outside the program makes it more flexible and easier to amend the text at any time.

A document may consist of a single piece of text, or it may be composed of a number of smaller clauses and paragraphs joined together into a single document as it is output.

Each document - or section of text which is to be included in another document - is created as an item on a file. The individual lines (or attributes) of the item comprise:

1) The text which is to appear in the document, and

2) Commands which control the format and the appearance of the document.

Runoff is a tool for *outputting* the document. It does not include an input facility, and one of the standard editors must be used for this activity to create the text item for Runoff.

We only have space for a brief summary of Runoff and its features. A full description of Runoff can be found in the the reference literature for your system and in the *MB-Guide to Runoff*.

A typical item, may look like this:

Input text
```
.* Demo document
 This is typical of the format of a Runoff document. It is
made up of lines of text and (optionally) Runoff command
lines. Some commonly-encountered Runoff commands are:
.INDENT MARGIN 5
.NOFILL

_.NOJUSTIFY
_.FILL
_.INDENT MARGIN
_.INDEX
_.JUSTIFY
_.NOFILL
.INDENT MARGIN -5
.FILL
.JUSTIFY
 All Runoff commands begin with a dot in the first position.
Any line which does not begin with a dot is assumed to be a
line of text for inclusion in the document.
 Unless you include Runoff commands to indicate otherwise,
Runoff will automatically:
```

```
.INDENT MARGIN 5
.PARAGRAPH -2
 * fill up the lines to the required length,
 * pad the lines with spaces so as to justify the right
margin (that is, make the right margin even).
 * capitalise the first letter of each sentence.
 * indent the first line of each paragraph by five spaces.
 * produce 60-character long lines.
 * produce 23 lines per page for a document which is
displayed on the screen (or 60 lines per page for printed
documents), taking these details from your terminal
characteristics.
.INDENT MARGIN -5
.PARAGRAPH 0
 Runoff will also put two blanks between sentences.
 The lines of text may be as long or as short as necessary.
If the lines of text follow a .FILL command, then the text
will be rearranged so as to fill up the line.
 text which follows a .NOFILL command (such as the list
above) will be left exactly as it was typed in by the author.
 If a line starts with a space, then Runoff interprets this
as a new paragraph.
```

This document would be created as an item on a file - using one of the editors, using a command such as:

```
EDIT MY.TEXT SAMPLE
```

When this item is processed with a command such as:

```
RUNOFF MY.TEXT SAMPLE
```

to display the document, or:

```
RUNOFF MY.TEXT SAMPLE (P
```

to print the document, then the final document would look like this:

Output document

```
     This is typical of the format of a Runoff document.  It
is made up of lines of text and (optionally) Runoff command
lines.  Some commonly-encountered Runoff commands are:

     .NOJUSTIFY
     .FILL
     .INDENT MARGIN
     .INDEX
     .JUSTIFY
     .NOFILL

     All Runoff  commands begin  with a  dot  in  the  first
position.  Any  line  which does not  begin with  a  dot  is
assumed to be a line of text for inclusion in the document.

     Unless you  include  Runoff  commands  to  indicate
otherwise, Runoff will automatically:

   * Fill up the lines to the required length,
```

* Pad the lines with spaces so as to justify the right margin (that is, make the right margin even).

* Capitalise the first letter of each sentence.

* Indent the first line of each paragraph by five spaces.

* Produce 60-character long lines.

* Produce 23 lines per page for a document which is displayed on the screen (or 60 lines per page for printed documents), taking these details from your terminal characteristics.

Runoff will also put two blanks between sentences.

The lines of text may be as long or as short as necessary. If the lines of text follow a .FILL command, then the text will be rearranged so as to fill up the line.

Text which follows a .NOFILL command (such as the list above) will be left exactly as it was typed in by the author.

If a line starts with a space, then Runoff interprets this as a new paragraph.

Obviously, Runoff is not a *wysiwyg* - what you see is what you get - system and the actual appearance of the document is only seen when it is finally output to the screen or printed on the printer. This fact, together with the requirement that the author (and/or typist) must be familiar with the Pick editor - or similar software - in order to create and maintain the document, means that Runoff is not a particularly user-friendly tool.

Runoff is provided with the sole purpose of producing printed (or displayed) matter. If you have used more sophisticated word processing software you will miss:

* The ability to handle diagrams and illustrations.

  As with the Pick system in general, Runoff does not handle graphics and has no facilities for including artwork and diagrams within the text, beyond simple boxes and tables. If required, these must be added later.

* The ability to use a range of fonts and type-faces.

  Runoff has standard facilities for using bold-face and underlining, but any other effects - such as italics - must be implemented by including the required printer-codes within the text. But since Runoff regards these codes as a part of the text, their use may upset the line format.

But it does seem a pity that so many people who, with a little effort, could learn to use Runoff ignore it completely

in favour of other cosmetically more attractive pieces of word processing software, thereby wasting a perfectly usable software tool.

Indeed, Runoff has one decided advantage over a great many pieces of more-sophisticated word processing software: it can use one basic document in a great many other documents. This simplifies the maintenance and production of documents such as contracts and specifications, which are invariably built up from a selection of standard paragraphs and clauses. With Runoff, any changes made to the basic document or clause are automatically reflected when the master document is next printed.

9.1 Creating / changing Runoff documents

The Runoff software does not include any means of creating and/or changing your documents. Runoff is solely concerned with the *output* of the formatted documents, displaying them on the screen or printing them on a printer.

The contents of the document must be maintained by the Pick line-editor or some other proprietary software which may be available to allow you to amend and correct your Runoff documents.

Each Runoff document - or section which is to be used in another Runoff document - is created as an item on a file. With each file, each item is uniquely identified by its item-id. The illustrations in the previous section show how each item comprises a number of lines (attributes), and each line is either a Runoff command (if it starts with a full stop in the first position) or a line of text.

9.2 RUNOFF command

The RUNOFF command is used to produce a displayed or a printed copy of a document. The command is typed in directly at TCL and the general format of the command is:

```
RUNOFF {DICT} filename itemlist {(options}
```

where *filename* is the name of file holding the document(s) which are to be output, and *itemlist* is a sequence of one or more item-ids for documents on the file and these are specified in the order in which they are to appear in the output document;

The *options* allow you to change the normal Runoff processing in some way. For example, you might use:

P to output the document to the printer.

S to suppress the boldface and underline effects.

U to output the entire document in UPPER-CASE.

Some examples of RUNOFF commands might be:

```
RUNOFF MB.TEXT SAMPLE1
```

```
RUNOFF MB.TEXT SAMPLE1 (P
RUNOFF MB.TEXT CHAP1 CHAP2 CHAP3
```

10 Spooler

The Pick system is a multiuser system and there may be several people using the same computer at the same time. Any - or all - of these users may ask for a printed report to be produced simply by issuing an Access sentence such as:

```
SORT STOCK BY PRICE DESCRIPTION TOTAL VALUE LPTR
```

with the LPTR modifier, or a TCL command such as:

```
COPY DICT STOCK * (P
```

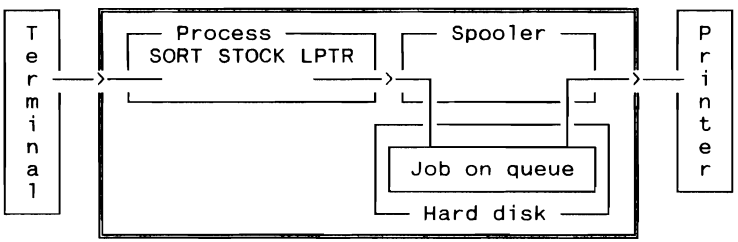
with the P option, or by executing a Basic program which uses the:

```
PRINTER ON
```

statement to print a report or produce some other sort of listing. If all the users were to ask for a report at the same time, there could be problems unless the system has some means of resolving the competition for the printer resources.

The *spooler* is a standard part of the Pick system and controls the production and output of all the reports and other output sent to the printer by the various users.

Whenever you perform a task which produces printed output, the spooler collects the output - line by line - as it is produced and stores this on a temporary disk file until the process finishes. When the process is complete your terminal is released and is then free to perform other tasks. In the meantime, the spooler takes care of the output, holding the various reports in a queue until the printer is free and your report can be printed. When the report has been output and no longer required, the temporary disk file is deleted.

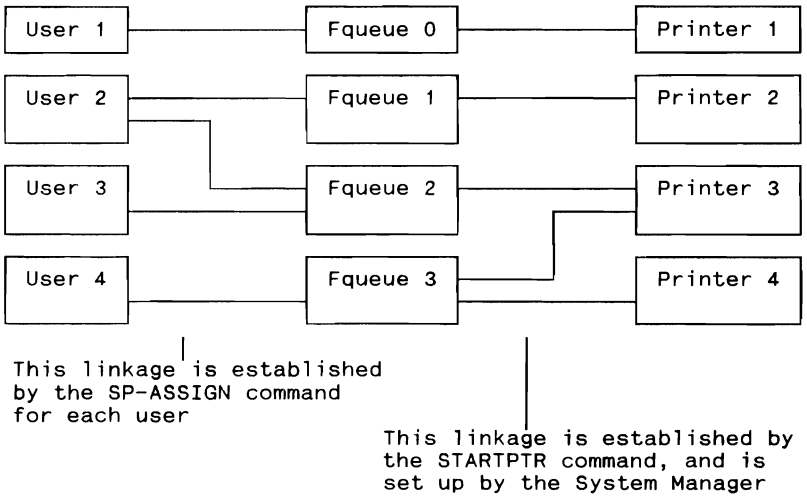


If there were no spooler, whenever you sent a report to the printer, you would have to wait until your entire job had finished and all the reports were completely printed before your terminal would be released and you could continue with other work.

10.1 Users - form-queues - printers

The diagram below shows the linkage between the users and the printer(s) which are available on the system. The linkage consists of the *form-queues*. There may be any number

of users, any number of form-queues and any number of printers. Each form-queue is identified by name, and each printer is identified by number.



The first part of the linkage is that between the users and the form-queues:

* A user may send his/her print jobs to just one form-queue (user 1 is linked to the form-queue 0 only). Unless they issue an SP-ASSIGN command to change this linkage, all users are connected to the form-queue 0.

* A user may be linked to several queues (User 2 is linked to Form-queue 1 and Form-queue 2).

* Several users may be linked to the same queue (user 2 and user 3 are both linked to Form-queue 2).

* This association is set up by the SP-ASSIGN command which is issued by each user. We shall discuss this later.

The second part of the linkage is between the form-queues and the printers:

* A form-queue may be linked to just one printer (the form-queue 0 is linked to Printer 1 only).

* A form-queue may be linked to several printers (Form-queue 3 is linked to Printer 3 and Printer 4).

  If a job is sent to form-queue 3, it will be printed on whichever of printers 3 and 4 is free.

* Several form-queues may be linked to the same printer (Form-queue 2 and Form-queue 3 are both linked to Printer 3).

* This association, set up by STARTPTR commands issued by the System Manager, is system-wide.

10.2 Why do you need to use the spooler?

We have seen that the spooler is necessary to handle the conflicting demands when several users are attempting to use the same printer (or printers) at the same time.

The spooler also handles the *de-spooling* of your reports after the process - the Access process, the TCL command or the Basic program which produced them - has terminated, controlling the reports as they are sent to the printer. This means that, once your report has been sent to the Pick spooler, your terminal is free for you to carry on with some other work. Some systems tie up your terminal from the moment that you issue the request to produce a report right until the very last line has been printed.

10.3 When do you use the spooler?

If you only want:

* To print a single copy of your output,

* On the main printer,

* As soon as the printer is free,

then you do not need to concern yourself with the spooler. You can simply carry out the work which produces the reports and these will be collected by the spooler and printed when your turn comes in the queue.

However, the spooler is much more useful than this and there are simple facilities which allow you to change this state of affairs:

* You may assign your report(s) to the queue associated with any of the printers which are available on your system.

* It is possible for a single program to produce several different reports: a payroll program might produce pay cheques, pay slips, departmental returns, and so on. In such cases, the spooler will allow you to specify that each of the reports be directed to a different printer, or some of the reports be directed to one printer, and other reports to another.

* You may specify that a sequence of reports be collected together and output as a single report.

* You may ask for your reports to be dumped to backing storage diskette/tape for storage and printing when required.

* You may ask that the first part of a report be printed first, allowing the operator to line up any special

pre-printed stationery such as invoices or cheques.

* You may request the spooler to suppress the output.

* You may request the spooler to hold the report on the queue after the job is completed.

When a report is held on one of the spooler queues, you may:

* Inspect it - to check the results before printing.

* Delete it - the results may be wrong.

* Move it to another queue - from where it will be output to another printer.

* Change the number of copies - you can produce several copies of a report without repeating the request.

* Hold the report for printing overnight - it may be too large to print during the day.

* Hold the report until you have had a chance to load and line-up the stationery on to the printer - you may want to print on cheques or special stationery.

* Copy the report and save it as an item on a file - you may want to incorporate an output report into some other documentation or save it for later use.

For these reasons, it is worth thinking about what the spooler can do for you.

10.4 Spooler commands

The spooler is a standard piece of software which controls all output which the users send to the printers connected to the Pick system. This subject is covered separately in the *MB-Guide to the Spooler.*

The fundamental TCL commands associated with the spooler include:

```
:STARTSPOOLER
LISTABS
LISTPEQS
LISTPTR
SP-ASSIGN
SP-CLOSE
SP-EDIT
SP-KILL
SP-OPEN
SP-STATUS
SP-TAPEOUT
STARTPTR
STARTSPOOLER
STOPPTR
```

Because some users find it difficult to remember and use all these commands, many implementations have front-end menu

routines which allow the user to select the work which is to be done, and the routines then invoke the appropriate commands from the above list.

The reference literature for your own system will give full details of this topic.

11 Operations

Because of the nature of the Pick system, it is not necessary to have a full-time operator in the traditional sense of someone who is responsible for running jobs, mounting tapes and diskettes, and so on; the individual users are themselves the operators. Nevertheless, there is a need for someone with responsibility for performing a number of domestic and control tasks. The actual title of the person (or persons) responsible for these duties depends upon the organisation; it may be *Operations Manager*, *System Manager*, *Chief programmer* or *Operator*. In this *MB-Guide*, we shall use the term *System Manager*.

Activities to be performed regularly as part of a standard rota are:

* Switching the machine on and off.

* Housekeeping activities and keeping the place tidy: *daily*.

* Checking and setting the current time and date on the system clock/calendar: *daily*.

* Cleaning the equipment - especially the terminals, the printer(s) and the read/write heads of the diskette/tape units: *weekly*.

* File-save routines: *daily* or more frequently if there is much activity on the files.

* File-restore routines: *weekly* or monthly, according to the fragmentation of the disk space.

* Producing and monitoring the file statistics: this can be done after each file-save, but it may only be necessary to save the last few copies of the report.

* Monitoring the size and shape of the files on the various accounts, using the statistics from the most recent file-save, reporting any notable situations and recommending that specific files be reorganised: *weekly*.

* Monitor the standard accounts, the standard files and their contents: *monthly*.

* Monitor the ACC file and account usage: *weekly* or more frequently according to the amount of data which is logged.

* Verifying the ABS frames: *monthly*.

* Producing reports on the standard files.

* Monitor the error log file and report any problems to the system engineer: *weekly*.

* Liaising with the maintenance engineer, arranging the periodic preventive maintenance visits and reporting any

system errors such as GFEs: *monthly*.

* Additional activities to be performed as required or as necessary:

  + Account-save.
  + Selective restore.
  + Create-account / delete account.
  + Archiving.
  + Trouble-shooting Activities:

    - Restoring the ABS frames,
    - Fixing group format errors.
    - Performing coldstart and warmstart.

These topics are covered in detail in other booklets in the *MB-Guide* series.

12 Using the system

Now let's us look at the operational aspects of using the Pick system.

Most of your work will be carried out using a terminal equipped with a screen and a keyboard.

12.1 The screen

The screen serves two functions:

1) To display all the information as you type it in. This allows you to check what you have typed.

2) To display the output produced by the programs and other software which you use.

Whenever you use the computer, you should ensure that:

* You can see and read the screen comfortably and without any strain,

* The screen is well positioned. You may be able to adjust the height and angle of the screen to suit you.

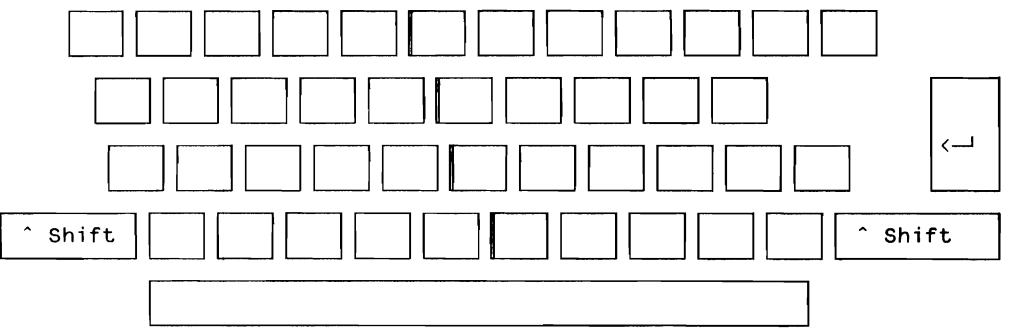
* That the brightness and contrast are comfortable. There will be one or more knobs for you to adjust the screen image.

12.2 The keyboard

The keyboard is the main method by which you will pass information and commands to the computer.

You don't have to be a skilled typist, but you will find it easier to use the system when you have become familiar with the layout of the keyboard.

The actual layout of the keyboard depends upon the manufacturer and the model which you are using. Typically, it looks like this:



If you are new to keyboards, you should pay particular

attention to the following parts of the keyboard:

* The main body of the keyboard containing the letters and, along the top, the numbers 1 to 0 and the special characters ! to +

* The space bar along the bottom of the keyboard.

* The <Return> or <Enter> key.

* The <Backspace> key. This may be marked BKSP or BACK SPACE or it may simply have a left-pointing arrow. You will use this key to correct any mistakes by deleting the last character which you typed in. As you press the <Backspace> key, the characters will be erased one by one from the screen display.

  You must make such corrections before you have pressed the <Return> key to send the information to the computer.

* The letters and numbers keys.

* The <Shift> key. There are two identical <Shift> keys at each side of the keyboard. These are used exactly like the shift key on a normal typewriter to select the upper choice from those keys which show two characters.

If you have used a typewriter keyboard before, you may recall that:

* The letters may be entered in UPPER-CASE or lower-case.

* If the <Caps Lock> is set on, and you press the [A] key, you will send the capital letter *A* to the computer.

* If the <Caps Lock> is not set on, and you press the [A] key, you will send the small letter *a* to the computer.

* If the <Caps Lock> is set on, and you press the <Shift> key and the [A] key, you will send the capital letter *a* to the computer.

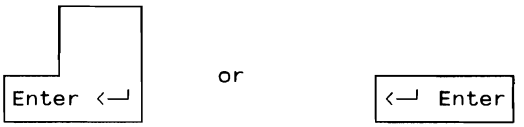
* If the <Caps Lock> is not set on, and you press the <Shift> key and the [A] key, you will send the small letter *A* to the computer.

* If you press the [! 1] key on the top row, you will send the number *1* to the computer.

* If you press the <Shift> key and the [! 1] key on the top row, you will send the *!* character to the computer.

* When you have typed all your information, you will press the <Return> key to send all the information to the computer.

The key which we have called <Return> in the text may look like the large key shown to the right of the keyboard in the diagram above, or it may be:



In addition to the numbers which are shown on the top row of the keyboard, there may also be a *numeric key pad.* This also allows you to enter the numbers and certain special characters, such as * and . The numeric key pad is only effective when the <Num Lock> key is set on.

Your system may also use:

* The <Esc> key. The escape key is used by some programs so that you can indicate that you want to finish or abandon the current process.

* The function keys along the top of the keyboard,

* The cursor control keys. These are four keys, each with an arrow pointing north, south, east or west.

* The <Ctrl> key. The control key is used to give a special meaning to the ordinary keys of the keyboard. You will always be told when and where to use these special control characters.

  For example, if you are told to press control-M or:

  <Ctrl> M

  you will hold down the <Ctrl> key and, whilst holding this down, press the M key at the same time.

You should only use this last set of keys when you are instructed to do so.

There may be other keys marked with names such as:

DEL
DELETE
END
HOME
INSERT
Ins
PAGE DOWN
PgDn
PAGE UP
PgUp
PAUSE
PRINT SCREEN
SCROLL LOCK

Do not use these last keys unless you are specifically instructed to do so.

12.3 Typing errors

You can correct typing errors by using the <Backspace> key to go back and correct your mistakes *before* you press the <Return> or the <Enter> key. However, there will be almost certainly be times when you type in the wrong information. What happens then?

In many cases, the software may recognise that you have made a mistake and will reject the information after displaying some suitable message. This is possible when the computer has been instructed (by the standard software, by the programmer or the analyst) what it is to accept. Thus, if you are typing in an amount of money and you mean to enter the sum:

```
123.50
```

then it is fairly easy for the programmer to arrange for the software to reject amounts such as:

```
THE CAT SAT ON THE MAT
SUNDAY
IZ3.5O
```

and even:

```
-123.50
```

may be unacceptable at that point. In the same manner, the programs will probably reject invalid dates such as:

```
29 FEB 1991
39 JAN 1991
-10 DEC 1990
```

But the software may be unable to detect that you meant:

```
123.50
```

when you actually typed:

```
132.50
```

or that you wanted:

```
21 JAN 1991
```

when you actually typed:

```
12 JAN 1991
```

Some application systems may give you an opportunity to change the information which you have entered before it is finally accepted by the computer. For example, the system may ask you:

```
OK? ENTER Y TO PROCEED OR LINE-NUMBER TO CHANGE
```

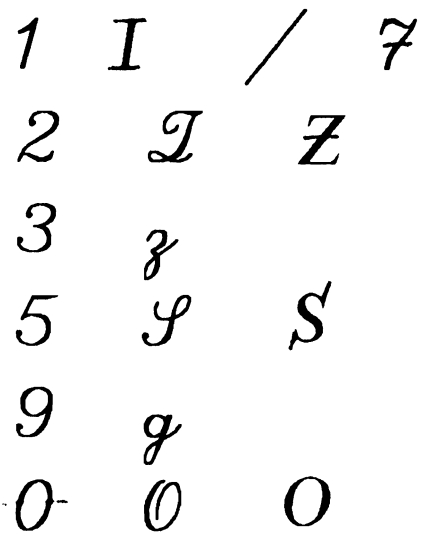
and you can then correct the offending information.

If you do make a mistake which you cannot (or did not) correct, then you should make a written note of what, when and where the mistake occurred and then report the matter to your supervisor.

12.4 Reading and writing

It is particularly easy to make mistakes when you are reading information from handwritten documents and typing this into the computer system.

For this reason, some people prefer to write special forms of those letters and numbers which are likely to be confused. Some of these are shown below.



In their handwritten form, these figures and letters are usually distinguished by having:

* An extra horizontal line as with the figure 0, the upper-case letter Z and the figure 7, or

* An exaggerated serif, as with the upper-case letters I and S, or

* An exaggerated tail, as with the lower-case letters g and z. It is most likely that the alphabetic information which you encounter will be in upper-case letters.

13 Switching on

There are one or two simple points to remember when you start to use your computer terminal:

1) Switch on the power at the electricity socket.

2) Switch on the terminal.

3) Check the brightness and contrast.

4) Press the <Return> or <Enter> key.

The system should then respond with a message such as:

LOGON PLEASE

If the computer does not respond with a message such as this, press the <Return> or <Enter> key again. If it still fails to react, then:

5) Check that the power cable is connected to the back of the terminal.

and:

6) Check that the computer cable is connected to the back of the terminal.

If all the connections are satisfactory but there is no response from the computer, then you should call your supervisor or the System Manager.

13.1 Logging on

In order to identify you and make the necessary files available to you, the system needs to know which account you wish to use. You do this by entering the name of your account. This process is known as *logging on to the computer.*

To log on to the computer, you need to know:

* The name of the account which you want to use,

and you may also be asked for:

* The password for that account.

If the system reacts by displaying:

LOGON PLEASE

and you make too many unsuccessful attempts to enter the correct logon code, the system will lock you out. To reactivate the terminal, you must:

a) Type any key, and then

b) Press the <Return> key (or the <Enter> key) twice.

The system should then ask you to LOGON PLEASE as before.

13.2 Account names

An account is simply a working environment in which you have access to all the files which you need, but in which you cannot reach those that you do not need.

In order to identify you and make the necessary files available to you, the system needs to know which account you wish to use. You do this by entering the name of your account. This process is known as *logging on to the computer.* We shall look at this later.

Write down the name of the account which you will use and make a note of the sort of work which you will carry out when you log on to that account. If there are several accounts available to you, make a note for each one.

Account(s) available to you

If you have forgotten the name of your account, you must ask your System Manager.

Any number of people may be using the computer at the same time, and any or all of these may be logged on to the same account. When several people are logged on to the same account, each of these people may be working quite independently of the others, or they may be using the same programs and processing data on the same files.

13.3 Passwords

For security reasons, some accounts may be protected by a *password. You must never write down the password* which you will use to log on to your accounts. The password will change from time to time and it will jeopardise the security of your system if an unauthorised person knows your account name and the password and is thereby able log on to your account.

If you have forgotten the password for your account, or if it has been changed whilst you were on holiday, you must ask your System Manager. Your System Manager will also change your password from time to time and he/she will inform you of the new password which you will use to log on to your account.

When you switch on a terminal which is connected to the system, there will be displayed a message inviting you to:

LOGON PLEASE:

and before you can do any work whatsoever, you must *log on* to identify yourself and indicate which account you are going to use. You do this simply by typing in the *name* of the account you wish to use.

If you do not know - or have forgotten - the name of your account, then you cannot log on to the system.

If your account is locked by means of a *password*, then you will be invited to enter this password before being allowed to proceed.

Beyond this standard Pick security, you may be using an application system which makes further checks on your identity and authorization before you are allowed to use the system.

When you have successfully logged on to your account, you can then start your work.

13.4 When you have logged on

When you log on to the system, any one of several things may happen:

1) You may be thrown straight into the business system for your organisation.

2) There may be a request for some further identification,

3) A series of menus may be displayed from which you will select the work which you wish to carry out.

4) The computer may simply display the:

```
>
```

symbol and wait for you to issue a command. When the system is at this stage, it is said to be *at TCL*. This means that the system is waiting for you to type in an instruction in the Terminal Command Language, TCL. We shall look at the TCL language later.

If you are using some versions of the system, these may display the colon:

```
:
```

symbol instead of the > and wait for you to issue a TCL command.

Make a note of what happens when you log on to your own system. If there are several accounts available to you, make a note of what happens for each one.

Account name
When you log on ...

13.5 Typing in the commands

When any information is to be entered at the keyboard - whether it is a TCL command or an Access sentence - there are a number of standard conventions which are designed to make your life easier:

<Ctrl> R to redisplay the sentence which you are currently typing in;

<Ctrl> W to backspace the last *word* of the sentence which you are currently typing in;

<Ctrl> X to ignore the sentence which you are currently typing in and start again. This avoids your backspacing to the beginning of the line.

When you are producing a long Access report and this halts at the bottom of the page waiting for you to press <Return>, you may terminate the report by entering:

<Ctrl> X

Instead of <Return>. The sequence:

<Ctrl> E

may also be acceptable on some implementations.

When you are entering any information at the keyboard, it does not matter if you type more than 80 characters (the width of the screen), the cursor will drop on to the following line and you can carry on typing. You may enter up to 140 characters at one time. However, if you attempt to type more than 140 characters, then the 140th character will terminate that input process, just as if you had pressed the <Return> key.

14 In difficulties

There may be occasions when you get into a situation where:

1) The work you are doing gets into trouble and stops with the system displaying something like:

```
I123
*
```

or it might display something like:

```
I 123.45
!
```

2) The computer may appear to be doing nothing, or it may seem to be a loop and doing the same thing over and over again indefinitely. In this case, your supervisor or the System Manager may tell you to press the:

<Break>

key. The system will then respond with one of the two sequences shown above.

Whenever the computer displays one of the sequences shown above, you may enter any of the following commands:

<Return>
used on its own, the <Return> key has no effect, except to repeat the * or the ! prompt.

<Line feed>
instructs the system to continue the processing, if possible. This is identical to the G command described below.

<Ctrl> J
instructs the system to continue the processing, if possible. This is identical to the <Line feed> command described above, and is valuable if you are using a terminal with no <Line feed> key.

END<Return>
instructs the system to terminate the processing and return to TCL or, if the account automatically invokes a Proc, to this logon Proc.

END<Line feed>
On some implementations, this will terminate the current activity and return the processing to TCL or, if the activity was invoked from a Proc, to the calling Proc.

OFF<Return>
OFF<Line feed>
instructs the system to terminate the current activity and log off the account.

G<Return>
G<Line feed>

instructs the system to continue the processing, if possible. This is identical to the <Line feed> command described above, and is valuable if you are using a terminal with no <Line feed> key.

P<Return>
instructs the system to switch the terminal print output function ON or OFF, and suppress the display to the terminal screen.

You should always consult your supervisor or the System Manager before you use any of these commands, otherwise you may lose some of the data or you may undo a lot of work which you or someone has performed.

The part of the system which is in control when either of the sequences:

```
I123
*
```

or:

```
I 123.45
!
```

is displayed is known as the *debugger*. It is intended to allow a technical programmer to debug or solve the problem which has occurred. So, if you want to report such a fault to someone, you will tell them that:

"... it has gone into the Basic debugger ..."

in the first case, where the asterisk is displayed, or:

"... it has gone into the system debugger ..."

in the second case, where the exclamation mark is displayed.

15 Logging off

When you have finished all your work and no longer need to use the computer system, you will log off. This instructs the computer to ignore you and your terminal until you (or someone else) logs on again.

If you are working within an application system, then there may be some formal way of finishing your processing and logging off.

If you are using TCL, then you will log off by entering the:

```
OFF
```

command.

You may then switch off your terminal, or you may wish to log on again.

If you simply want to log on to another account, you may do this directly by entering a command such as:

```
LOGTO WAGES
```

which will terminate your work on the current account and then log you on to the WAGES account. If there is a password on the WAGES account, you will be asked to enter this.

15.1 Switching off

If you have finished using your terminal - you may have finished work for the time being, you may be going to lunch, or you may be going home at the end of the day - then you will:

1) Log off, as we have just described.

2) Switch off the terminal.

3) Switch off the power at the wall socket.

Your supervisor or the System Manager will tell you of any other security measures which apply to you.

16 Some jargon

It will be useful if you become familiar with the words which are used in this MB-Guide and elsewhere in the literature about the computer system. The most important words and concepts are summarised here.

Access
: The enquiry language which is available on the Pick system and which allows you to make enquiries and to produce reports by typing in an English sentence. The language is also known by other names such as English, Info/Access and Recall.

Account
: A set of related files which are grouped together and accessible to anyone who logs on to that account.

  It is usual to group all such files together into one *account*. When you *log on* to the computer system, you will specify the name of the account which you want to use; this will give you access to the files on that account.

Attribute
: The Pick terminology for a field or a data field of a record.

  Each field of a Pick data item is known as an attribute. Within the item, the attributes are separated by the attribute mark (ASCII character 254).

  Each attribute contains any number of *values* separated by value-marks.

  Each value contains any number of *subvalues* separated by subvalue-marks.

  Attributes, values, and subvalues are of variable length, each being terminated by the appropriate field-separator.

  The various data fields - the attributes, values, and subvalues - are identified by their sequential position within the item.

  A field which contains no data is known as a null field. You should distinguish between a null field (which contains no data whatsoever) and a field which contains only spaces. When you are entering information at the keyboard, you will normally represent a null field just by hitting the <Return> key.

  Null fields are represented only by the associated (following) field-separator - attribute-mark, value-mark, or subvalue-mark.

Basic
The language which is used to write programs for the computer. There are many such programming languages available. The standard Pick system can handle programs written in the Basic language and the Proc language.

Byte
A unit of storage (comprising 8 *bits*). Each character (each letter, each digit, each comma, each space, each full stop, and so on) occupies one byte of storage in the computer's memory.

File
A collection of items holding data of a similar nature. This is identical to the standard use of the term file within conventional data processing.

A file may contain any number of items (or records).

Frame
The operating system moves its information around in chunks of 512 bytes. Such a 512-byte chunk is known as a frame. In each frame, the first 12 bytes are used by the operating system for control purposes, the remaining 500 bytes are used to store data. On some systems, the frames may be 1024 bytes (24 control bytes plus 1000 data bytes), 2048 bytes (48 control bytes plus 2000 data bytes) or more in extent. A frame is equivalent to a sector of disk.

Hardware
The general term used to describe all the mechanical devices associated with a computer system. The hardware includes: the central processing unit; the terminal; the printers; the connecting wires and circuits. See also SOFTWARE.

Item
The Pick term for a data record. Each file consists of any number of items. Each item consists of the *item-id* which identifies the item, followed by none, one or more *attributes* separated by attribute-marks.

Item-id
The Pick term for a record *key*. The item-id uniquely identifies each item on a Pick file.

Log on
To identify yourself to the computer system; this tells the operating system the name of the account which you wish to use.

Operating system
A program which controls all aspects of the computer's work. The operating system is active all the time and takes care of: identifying and accepting/rejecting users as they log on; the printed output which is sent to the printer by the various users; accepting the users' instructions and commands; carrying out the work invoked by the users.

We are concerned with the Pick system. Other operating systems are available: Unix DOS; CP/M.

Password
A second piece of information (additional to the account name) which may be required when you log on to the computer system. For security purposes, the password may be changed from time to time.

Proc
A sequence of instructions (such as an Access sentence or a TCL command and possibly with other instructions in the Proc language), which has been saved under a single name so that it can easily be recalled in future.

The name is also used to refer to the language which is used to write such processing routines. The language is peculiar to the Pick system.

Program
A sequence of instructions which are submitted to the computer and which instructs the computer exactly how a certain task is to be performed.

Report
Any list of information which is displayed on your screen or printed on the printer.

Software
Any computer program, although the name is usually reserved for programs which are supplied by someone outside your organisation. See also HARDWARE.

Spooler
That part of the system software which controls the output which the users send to the printer(s).

I n d e x

MB-Guide beginners guides

The following titles are available in the *MB-Guide* series:

* Access definitions & dictionaries
* Access sentences
* Advanced Pick
* Advanced Pick: AP/DOS
* Advanced Pick: AP/NATIVE
* Basic language
* Basic programming topics: 1
* Basic programming topics: 2
* Basic symbolic debugger
* CompuSheet+
* Creating and using Procs
* DOS for Pick users
* Development standards
* ENGLISH
* Error Messages
* File design
* File-save & file-restore
* Files: file sizing tables
* Files: monitoring & sizing
* Group format errors
* Jet word processing
* MB-EDITOR : Screen editor
* Operations & systems management
* Pick fundamentals
* Pick on the PC
* Pick terminology
* Producing training courses
* Program design
* Reality
* Runoff text processing
* Security
* Spooler
* System debugger
* System design
* The Pick system
* Using Pick
* Using backing storage
* Using the Jet editor
* Using the Pick editor
* uniVerse for Pick users

In preparation

* Accu/Plot
* Basic programming topics: 3
* Mathematics for computing
* Pick: reference tables
* Programming in C
* SQL
* System health check
* commercial computing

MB-Guides

The booklets in the *MB-Guide* series cover a range of fundamental topics of interest to users and those responsible for developing, implementing and running Pick systems.

Each *MB-Guide* deals with a specific aspect of the operating system and the booklets represent an economical introduction to the various topics and the whole series forms an integrated presentation of the subject matter.

The booklets are intended to be a working document and, for this reason, space is provided for the user's notes, and the reader is encouraged to amend the booklet so that it applies to his/her own system.

The series of *MB-Guides* is of special interest to new users, and is a convenient source of information for training organisations, software houses and others who are responsible for the instruction and support of their clients and staff in the fundamental aspects of the Pick system.

Malcolm Bull

Training and Consultancy Publications