Malcolm Bull
Training and Consultancy Publications

MB-Guide to

AP/DOS

Malcolm Bull

© Malcolm Bull

MB-Guide

to

AP/DOS

MB-Guide

to

AP/DOS

by

Malcolm Bull

(c) Malcolm Bull 1992

MALCOLM BULL Training and Consultancy Services

(c) MALCOLM BULL 1993

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

Telephone: 0484-713577

ISBN: 1 873283 60 1

Edition: 2.5
Updated: 13:07:93

No part of this publication may be photocopied, printed or otherwise reproduced, nor may it be stored in a retrieval system, nor may it be transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without prior written consent of Malcolm Bull Training and Consultancy Services. In the event of any copies being made without such consent or the foregoing restrictions being otherwise infringed without such consent, the purchaser shall be liable to pay to Malcolm Bull Training and Consultancy Services a sum not less than the purchase price for each copy made.

Whilst every care has been taken in the production of the materials, MALCOLM BULL assumes no liability with respect to the document nor to the use of the information presented therein.

The Pick system is a proprietary software product of Pick Systems, Irvine, California, USA. This publication contains material whose use is restricted to authorised users of the Pick system. Any other use of the descriptions and information contained herein is improper.

The use of the names PICK, OPEN ARCHITECTURE, ADVANCED PICK and all other trademarks and registered trademarks is gratefully acknowledged and respected.

Contents

P r e f a c e

Most of the *MB-Guide* beginner's guides have been dedicated to the R83 implementation of Pick, and together they cover most aspects of that version of the system. This *MB-Guide to AP/DOS* is concerned with the Advanced Pick implementation and specifically to the version which runs under the DOS - the Disk Operating System - on IBM PC and compatible machines.

This *MB-Guide* considers:

* The major features of the Advanced Pick implementation, highlighting

* The differences between R83 and Advanced Pick.

* The installation of Advanced Pick on an IBM PC or compatible.

Rather than repeat everything that has been said in the other *MB-Guides*, we have chosen to concentrate upon the *differences* between R83 Pick and Advanced Pick. In the main, these are completely new features or enhancements to existing features. A detailed description of all aspects of Advanced Pick can be found in Reference Manual.

The material will be of interest to those people who are moving from R83 and other implementations across to the *AP/DOS: Advanced Pick* implementation. The parallel booklet entitled the *MB-Guide to AP/NATIVE* presents the same material for those migrating to the native Advanced Pick implementation.

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

* Access definitions & dictionaries
* Access sentences
* DOS for Pick users
* Operations & systems management
* Pick fundamentals
* Pick on the PC: this presents a general introduction to the Pick system.
* Using Pick

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

* PICK1: Starting Pick
* PICK2: Pick systems management
* ACCESS1: Starting Access

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.

Introduction

The Pick system has gone through a number of revisions. The most popular implementation of Pick is that known as R83, release 83, and is sometimes known as *generic* Pick. Since its original release in the early 1980s, the release has gone through several versions and the current version is R83 version 3.1

All other implementations of the operating system (with the exception of those described below) are based upon the R83 release. In most cases, the individual licensees have added their own facilities.

Following the production of the R83 release, Pick Systems introduced a number of radical changes:

* File indexing
* Level pushing
* 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 are:

* An Update Processor to facilitate the creating and maintenance of items through the dictionary definitions.

* An Output Processor based upon the principles of Runoff and the Update Processor.

* 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 host operating systems such as DOS and Unix, as shown below.

Pick Systems major products are now the R83 release and the AP releases:

* AP/Native: this is a free-standing product running on computer systems based upon Intel 286 and later micro-processors.

* AP/DOS: this is a version of AP which runs as a program on computer systems running under DOS.

* AP/RS6000 AIX, AP/AT&T Unix, AP/SCO, AP/DGUX: these are versions of AP which run as a process on computer systems running under the various manifestations of the

Unix operating system.

These last implementations should not be confused with systems such as UniData and uniVerse which are Pick-like systems running on Unix-based platforms.

A standard version (currently 5.2) is offered on all these AP implementations. AP versions 6.0 and higher are available for the AP/Unix implementations and offer facilities which exploit the features of Unix and the C language.

What is AP/DOS

AP/DOS now makes the full features of the Advanced Pick implementation of the Pick system available as a software package running within a DOS partition on an IBM PC or compatible.

This means that:

* DOS commands can be issued at TCL or by way of a Basic EXECUTE statement.

* Your PC can now run a Pick application just as it would run Lotus 1-2-3, WordPerfect or any of the many other DOS packages.

* DOS files are accessible. The R83 implementation allowed you to transfer data between the Pick partition and the DOS partition, but DOS files were not directly accessible.

* Subroutines written in other languages - such as C - can be accessed from Basic.

Like a PC, the AP/DOS implementation is a one-user system, each PC working as a separate Pick system. However, data may be passed between machines over a Novell network.

2.1 R83 / AP differences

Rather than repeat everything that has been said about the R83 implementation in the other *MB-Guides*, we shall only discuss the differences between R83 Pick and *Advanced Pick*. In general, these are completely new features or enhancements to existing features. We shall look at the following major areas:

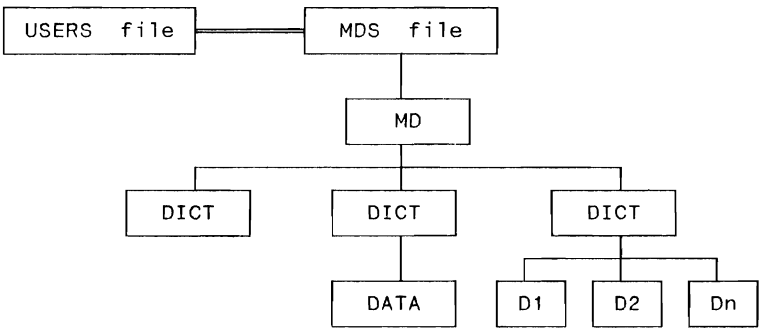
* File hierarchy and system organisation
* Accounts and files
* Q-pointers and pathnames
* Starting / leaving Advanced Pick
* Logging on / off ++
* Output Processor ++
* Keyboard functionality ++
* TCL and TCL commands
* Stacker ++
* Level pushing ++
* Phantom processes ++
* Macros ++
* Menus ++
* Transaction logging ++
* Incremental file-saves ++
* Update Processor ++
* Access and Access processing codes
* File indexing ++
* Basic
* The spooler

Those marked with the ++ symbol will be of special interest to anyone moving from R83 to Advanced Pick.

One important point is that Advanced Pick no longer actively supports Procs, indeed the Reference Manual makes no mention of them. For compatibility, however, the Proc processor is still available, and is identical to the version under R83. This means that all the Procs which your application system used under R83 will still work on OA and *Advanced Pick*, but almost all those standard Pick features and utilities which were previously implemented by Procs are now presented as Basic programs or macros. For end-users, the concept of Macros and Menus offers a simpler alternative to Procs when producing new systems.

File hierarchy and system organisation

The file hierarchy is essentially the same as on R83, except that some of the account/user information which was formerly held on the SYSTEM file has now been dispersed between the USERS files and the MDS file. As we shall see, the USERS file contains information about the individual users, and the MDS file contains information about the accounts and their MDs.



A file may have any one of the three structures shown in the diagram:

* A DICT only file,
* A DICT section and a single data section, or
* A DICT section and several data sections.

The USING clause is still available in Access sentences to use the DICT of one file with the data of another.

3.1 D-pointer items

D-pointers are encountered as account-definition items (on the MDS file) to define accounts, as file-definition items (on the MD of the appropriate account) to define file dictionaries, and as data-level identifiers (on the DICT section of the appropriate file) to define the data section(s) of the file.

The format of account-definition items and file-definition items (and also data-level identifier items) has been extended somewhat.

On Advanced Pick, the attribute-definition item has the following structure:

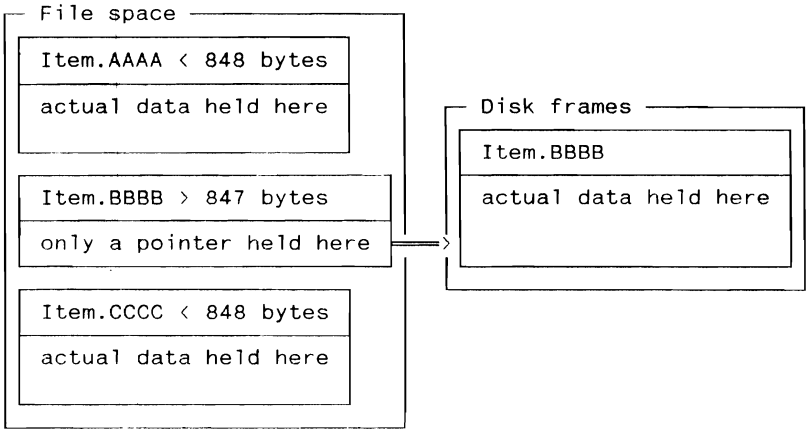
```
001 dictionary-code
002 attribute-count
003 substitute-header
004 structure
005 not used
006 not used
007 output-conversion
```

```
008 correlative
009 attribute-type
010 column-width
011 not used
012 not used
013 not used
014 input-conversion
015 macro
016 output-macro
017 description
018 not used
019 not used
020 hotkey.all
021 hotkey1
022 hotkey2
023 hotkey3
024 hotkey4
025 hotkey5
026 hotkey6
027 hotkey7
028 hotkey8
029 hotkey9
030 hotkey0
```

1) Attribute 1: this is the D-pointer which defines each DICT section and each data section. This consists of the letter D followed by suitable combinations of the following codes:

   L indicating that this file is to be the subject of *transaction logging*, and all amendments to this file are to be logged.

   P Short items are normally held in the file space, as on R83, but large items (848 bytes or more) are loaded directly into frames of disk space with only a *pointer* in the file space. This is illustrated in the diagram below and discussed again in the section dealing with file structure. The P code forces *all* items to be held in this pointer form. Organising a file in this manner, means that the process of finding any particular item is much faster (since the physical items on the file are only pointers of 8 bytes in size), but the retrieval of any item will always require at least two disk read operations (one or more to find the pointer, then a further one or more to retrieve the item from the frames of virtual memory).



S indicating that item-ids are to be case sensitive. Thus, on a file defined with this code, the keys *FRED*, *Fred* and *fred* are three distinct item-ids. This facility is necessary since all TCL commands (including those which specify an item-id) may be entered in upper- or lower-case.

X
Y are used as on previous versions to indicate that the file is to be ignored (X) or that the contents are to be ignored (Y).

Attributes 9, 14, 15 and 17 are used as described later in our discussion of attribute-definition items. All other attributes are used as previously (including the reallocation parameter in attribute 13).

9) Attribute 9: as on R83, this specifies the justification of the output data. Extended codes are available on *Advanced Pick*.

14) Attribute 14: this specifies the input conversion.

15) Attribute 15: this specifies the macro - a list of attribute-definition names which are to be used when zooming to another item with the Update Processor.

17) Attribute 17: this is the description - a free-format text field for any comments concerning the use and nature of the file/attribute.

20) Attributes 20 onwards: these contain the CALLs to any subroutines which are to be invoked by the hot-key sequences. We discuss hot-keys in a separate section.

3.2 System accounts

Each Advanced Pick system comes with a number of standard accounts and files already established:

* DM - data manager account.
* PA - personal assistant account. This offers basic office management facilities.
* QA - quality assurance account. This is a means of fault and error reporting.
* TUTOR account.

The most important of these is the DM account. This is the equivalent of the R83 SYSPROG account and is used for all the important operational tasks:

* Creating new users and accounts,
* Performing file-save and file-restore operations,
* Deleting accounts.

DM is only *privileged* by having the necessary verbs which other (non-privileged) accounts lack on their MDs.

3.3 Creating / deleting users

If the system has been set up appropriately, a new user can be created by entering:

```
?
```

at the request for a user-id when logging on.

The System Manager can also create a new user directly by means of the command:

```
UPDATE USERS username
```

and the Update Processor will ask for the following information:

Name	Enter the full name of the user
Address	Optional
Zip	Optional
Phone	Optional
Keys	Retrieval / update lock-codes
Password	See (1) below
Privilege	Enter one of SYS0 or SYS1 or SYS2
Options	Optional
Macro	See (2) below.

(1) If a password is specified at this stage, it should entered in the clear (unencrypted) form; the Update Processor will encrypt the password and display the encrypted form. The password may also be set by means of the TCL PASSWORD utility, as on R83.

(2) The macro and subsequent lines may contain a series of TCL commands which are to be executed when that user logs on. These commands serve the same purpose as the logon Proc in R83.

An existing user can be deleted by deleting the USERS item through the Update Processor by means of the <Ctrl> X O or XO

key sequence.

3.4 Creating / deleting accounts

A new account can be created in the normal manner, by means of a command of the form:

```
CREATE-ACCOUNT accountname
```

and the Update Processor will be invoked to create a new item on the MDS file. The user will be invited to enter the following information and when <Return> is entered after the final (REALLOCATION) input field, the new account will be created.

Type	This will be D
Modulo	37
Ret-lock	Optional
Upd-lock	Optional
Password	See (1) below
Syspriv	SYS2
Justification	L
Width	10
Reallocation	Optional or in the form (mod)

(1) If a password is specified at this stage, it should entered in the clear (unencrypted) form; the Update Processor will encrypt the password and display the encrypted form. The password may also be set by means of the TCL PASSWORD utility.

An account may also be created by invoking the Update Processor directly:

```
UPDATE MDS accountname
```

in which case, the following information will be requested:

Type	This will be D
Ret-lock	Optional
Upd-lock	Optional
Attribute-type	L

and the completed item finally filed by the <Ctrl> X F sequence.

An account password may be assigned and/or changed by means of the TCL PASSWORD utility.

Similarly, an account may be deleted by either of the forms:

```
DELETE-ACCOUNT accountname
```

or

```
UPDATE MDS accountname
```

followed by the <Ctrl> X O sequence.

In either case, the user will be asked whether he/she wants:

DISPLAY FILES BEFORE DELETING

DO YOU STILL WANT TO DELETE THE ACCOUNT

3.5 System files

Advanced Pick has several standard files holding information which is necessary to all users. Some of these are accounts in their own right (as the MDS file/account), some are files owned by the DM data manager account, whilst others may be individual files established for each user-account.

MDS			
DM account			
ABS ACCOUNTS BLOCK-CONVERT BP BULLETIN DEPT	DEVICES ERRORS FILE-OF-FILES FONTS FUNCKEYS GSYM	JOBS KEYBOARDS MESSAGES NEWAC PERSONNEL PIBS	POINTER-FILE TCL-STACK USERS WORDS
PA account			
BP CATEGORY CODES	COUNTRY ENTITY JOURNAL	LOCATION STATEMENT ZCF	
QA account			
BP.QA COMPONENT LST MANUF	MODEL O/S PRIORITY REQUESTS	RESULTS STATUS.CODES TP.CODES TPS	TPS.RUN TYPE
TUTOR account			
BK BP	UP.DOC UP.DOC.REPORT	UP.PRINT.CODE	

The most important of these standard files are:

* MDS: holds details of all the master dictionaries (accounts) on the system and is equivalent to the SYSTEM file on R83 systems.

* ABS: holds details of the ABS frames and ABS usage.

* ACCOUNTS: holds the accounting history information and is equivalent to the ACC file on R83 systems.

* BLOCK-CONVERT: holds information about the large block letters which are used when you display or print messages by means of the BLOCK-PRINT command.

* BP: holds all the support programs used by the operating system.

* DEVICES: holds details of all the devices recognized by the system.

* ERRORS: holds details of any errors detected by the system.

* FILE-OF-FILES: holds details of all the files on the system and is equivalent to the STAT-FILE on R83 systems. An item is added to this file whenever a new file is created and the file is reset during an account-restore or file-restore operation.

* JOBS: holds details of jobs handled by the phantom processor.

* MESSAGES file: holds the standard help- and error-messages used by the system and is equivalent to the ERRMSG file on R83 systems.

* PIBS: holds details of all the processes which are currently executing on the system. There is one item on the PIBS file for each port on the system, and the data recorded here is similar to that held on the ACC file on R83 systems. The item-ids are the port numbers (no leading zeroes), and the item holds details of the port, including the location, the terminal characteristics of the port (these are invoked when that port is logged on), the time, date and identity of the last user, and the time, date and identity of the last-used MD.

* POINTER-FILE: holds details of all the saved-lists which have been established on the system.

* TCL-STACK: holds the stacks of TCL commands issued by the users.

* USERS: holds details of all authorized users of the system and their user-ids.

3.6 File structure

The frame size on Advanced Pick is 1024 bytes on some implementations and 2048 bytes on others, and these frames are used exactly as on previous versions of the operating system.

The concept of separation is no longer used. Only a modulo is required to define any section of any file, and the number of frames in each group (the statistic previously known as the separation) is always 1.

The same hashing algorithm is used as on R83, and the accepted conventions for the choice of modulo (selecting a prime number, or at least, avoiding multiples of 2 and 5) still apply.

Let us look at the DUMP of a typical frame on Advanced Pick. We have shown only the first few bytes in this instance, the remainder of the frame is filled with rubbish not related to our file.

```
000 :........AAAA^this is item aaaa and it will be held:
050 : directly within the file space.^_........BBBB^002:
100 :268^__.....................................
```

Here we see:

* An 8-byte field: this is the item-length field and an indication of the nature of the item (whether it is direct or a pointer item) are held within a control string of 8 bytes before each item-id.

* The end-of-item marker (^_) and the end-of-data marker (^__) are used as before.

* Items such as AAAA of which the data is 847 bytes in length or shorter are held in the primary file space (as on previous versions). This figure of 847 bytes relates purely to the length of data in the item and excludes the field-length count, the item-id and the final end-of-item marker.

  If we look at this 8-byte field as a hexadecimal string, we find that it is of the form:

  00000000yyxx1000

  where xxyy is the hexadecimal length of the item (this represents the 8-byte field itself, plus the length of the item-id, plus one attribute-mark, plus the length of the true data and field separators, plus 1 for the final attribute mark); note the way in which the xxyy field is held. We should also mention that this value is held as an odd number (1 being added, if necessary). The 1 in the 13th digit indicates that this item is held in the file space.

* Longer items (848 bytes or more) such as BBBB are written directly to one or more frames of disk and only a pointer-item is held in the file space. If we dump the frame to which the pointer for item BBBB directs us (hex 2268 = decimal 8888), we see the full item:

  For items held in this manner, the 8-byte field would be:

  0000000011009000

  with the 9 indicating how the item is held.

```
          fid: 8888 : 0 0 0 0 ( 2268 : 0 0 0 0 )

000 :_BBBB^this is item BBBB and, since it exceeds 847 :
```

```
050 :bytes, it is held as a pointer in the file space. :
100 :   This pointer points to the frame where the true:
150 : data is held.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:
200 :xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:
:
850 :xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:
900 :xxxxxxxxxxxxxxxxxxxxx^__.....................:
950 :.............................................:
```

This example was produced on an Advanced Pick system which uses 1024-byte frames. The effect is similar on those systems which uses 2048-byte frames.

The most significant consequence of this is that there is now virtually no limit to the size of an item. The 32K restriction of R83 is now raised to 2 gigabytes under *Advanced Pick*.

Note that, for larger items, there will always be a requirement to perform two (or more) disk accesses to find a specific item: one to find the pointer and then a further one (or more) to find the item itself.

If the D-pointer for a file includes the P code in attribute 1, then all items - large and small - will be held as pointer items.

These points have some impact on the subject of file-sizing. The algorithm discussed in the *MB-Guide to files: monitoring and sizing* will use either of two values for the average size of the items: if data within the items is likely to be 847 bytes or less in length, then this figure should be used in the algorithm; if the data is likely to be 848 bytes or more (or it has been established with a DP pointer, as discussed in Section 3.1), and the item is to be represented by a pointer item, then a figure of 6 bytes (the size of the data in a pointer item) should be used in the algorithm.

3.7 Pathnames

Q-pointers may be used to access files on other accounts and can be set by means of the SET-FILE command. It is worth noting that the syntax of the SET-FILE command is slightly different:

```
SET-FILE account.name file.name {synonym.name}
```

where a synonym-name may be included to specify the name by which the Q-pointer is to be saved on the MD of this account. If this synonym-name is omitted, the name QFILE will be used, as on R83 systems.

Advanced Pick, offers the concept of *pathnames* in which a file-name can be specified in TCL commands and Access sentences in any of these forms:

aaaa,,
: to identify the MD of account *aaaa*.

aaaa,ffff,
to identify file *ffff* on account *aaaa*.

aaaa,ffff,dddd
to identify data section *dddd* of file *ffff* on account *aaaa*.

Note that two commas must always be specified, and the trailing comma(s) are essential in the first two instances.

Here are some typical uses of pathnames:

```
EDIT SYSPROG,, REBOOT
COPY TRAINING,STOCK, *
UPDATE PERSONNEL,STAFF,RETIRED 1234/AA
EDIT DICT TRAINING,STOCK, COLOUR
LIST TRAINING,STOCK, DESCR TOTAL VALUE
```

There is also the command:

STEAL-FILE *aaaa,ffff,*

which transfers the file-definition item for the file *ffff* from the MD of an account *aaaa* to the current MD.

3.8 POVF

The POVF command (and the synonyms OVERFLOW and OVF) outputs a summary of the space available. There are options:

A to output the FIDs and numbers of frames (this reproduces the normal R83 report);

B to output just the numbers of frames available;

n-m to include only those frames with FIDs in the specified range.

The system automatically sets aside a number of frames for use in emergencies when the system is running out of disk space. These reserve frames are shown in the POVF report. You can use the commands:

SET-OVF-RESERVE (n

if you wish to change the number of reserved frames, or even:

SET-OVF-RESERVE (o

to release them entirely for emergency use.

4 Installing AP/DOS

To install AP/DOS, you will need:

* the set of system diskettes and file diskettes for AP/DOS and

* your 12 digit *activation number.* This is supplied with the diskettes. Make a note of your activation number here:

Advanced Pick activation number

You will then:

1) Boot the DOS system,

2) Insert the Pick INSTALL diskette into drive A,

3) Enter the DOS command:

```
A:
```

4) Enter the command:

```
INSTALL
```

You should then follow the instructions displayed as AP/DOS is installed.

Having installed the software, you should then invoke AP/DOS:

5) Enter the DOS command:

```
PICK:
```

and then choose option

```
A
```

to load the ABS data, then repeat this and choose option

```
F
```

to load the Files. At each stage, you should follow the instructions which are displayed by AP/DOS.

AP/DOS is then ready to use.

Full installation instructions are supplied in the User's Guide which accompanies each set of AP/DOS diskettes.

4.1 DOS and AP/DOS - disk cache

AP/DOS will run under any of the current versions of DOS.

Because of the SMARTDRV disk caching software which is provided on MS-DOS version 5.0, you will benefit from using that version of DOS. This greatly increases the speed of AP/DOS by using expanded or extended memory to store data being read from the hard disk. To use SMARTDRV, you should add to your CONFIG.SYS file, a command of the form:

```
DEVICE=C:\DOS\SMARTDRV.SYS n
```

where *n* is the size (in Kb) of the cache which is to be used. This may be any integer within the range 256 to 2048. Full details of this command are given in the DOS 5.0 User's Guide and Reference manual.

If you have any other disk caching software, this may be used with AP/DOS.

5 Starting up AP/DOS

AP/DOS runs exactly like any other DOS application. To use AP/DOS you will:

1) Switch on your PC.

2) When you get the:

```
C:>
```

or similar DOS prompt, change to the directory where Pick is to be found by entering a command of the form:

```
CD\ddddd
```

Your System Manager will tell you the name of the directory. Make a note of the command(s) which you will use:

3) Issue the DOS command:

```
PICK /X
```

or:

```
PICKX
```

and this will invoke AP/DOs directly. Alternatively, you may issue the DOS command:

```
PICK
```

and at the prompt:

```
Options: A) ABS, F) Files, X) Execute, Q) Quit =
```

you should press:

```
X
```

but do not press <Return>, and then wait whilst system carries out a brief initialisation operation. You will then be invited to log on.

Whilst the system is performing its initialisation and setup, you should not touch the keyboard until the system invites you to enter your user-id.

Your System Manager may arrange for your AUTOEXEC.BAT file to include the necessary commands to change to the appropriate directory and issue the PICK /X command automatically. By doing this, you will be thrown straight into AP/DOS when you boot up your machine.

5.1 Logging on

Because of the structure of the Advanced Pick system, a user needs to supply *two* sets of information when logging on:

1) The user-id (and password, if there is one), and

2) The account-name (and password, if there is one).

So, in general, a user will first identify himself/herself as a user and then choose which (account) master dictionary he/she wishes to use.

The user-id is the item-id for that user as it is held on the USERS file. The information here specifies:

* The L/UPD and L/RET lock-codes for the user.
* The password for the user.
* The system privileges level for the user.
* The accounting options.

The account-name (or master dictionary name) is the item-id for the required master dictionary as it is held on MDS file. The information here specifies:

* The location and modulo of the master dictionary for the account.
* The L/UPD and L/RET lock-codes for the account.
* The password for the account.
* The system privileges level for the account.

A typical logging-on sequence for Advanced Pick will look something like this:

```
08:30:59  28 Jul 1993
Advanced Pick - Enter your PICK user id: aaaa,bbbb
master dictionary: xxxx,yyyy
```

where *aaaa* is the user's identity code (as on the USERS file), *bbbb* is the user's password, *xxxx* is the name of the account which is to be used (as on the MDS file), and *yyyy* is the password for that account.

Having logged on in this manner, a user may subsequently change to another master dictionary by means of the:

```
     LOGTO xxxx,yyyy
or:
     TO xxxx,yyyy
```

command without having to re-identify himself/herself again.

The use of passwords is optional in all cases, and you will only be required to enter the password if one has been applied to your user-id and/or the master dictionary. As on R83, the user-id and/or the account-name may be entered alone and any password will be accepted invisibly.

When you have logged on, the operating system will first look to attributes 12 onwards of your item on the USERS file and execute any TCL commands held there. This is the equivalent of the logon Proc on R83 Pick. When this has been done, you may be passed to TCL and the:

```
:
```

prompt will be displayed.

If the USERS item contains a LOGTO (or TO) command here, then this will automatically log that user to the specified account without asking for the name of the master dictionary required. This makes the logon procedure almost identical to that of R83 systems.

5.2 Logging off

When you have finished using the Pick system, you can take one of a number of courses of action:

1) You can log to another master dictionary. You will do this by means of one of the commands:

```
LOGTO xxxxx,yyyy
```

or

```
TO xxxxx,yyyyy
```

2) You can log off the system but leave the terminal free for any other user. You will do this by means of the command:

```
OFF
```

and then the Pick system will ask you for the Pick user-id and the master-dictionary name, as when you first log on.

3) You can leave Pick and return to DOS, as described in the following section.

The system will perform a brief wrap-up operation and then return you to the DOS:

```
C:>
```

prompt.

At this point, you may then use DOS in the normal manner or you may switch off your machine.

5.3 Leaving AP/DOS

As we saw in the previous section, you can leave Pick and return to DOS by logging to the DM master dictionary, by means of the command:

```
LOGTO DM
```

and then issuing the:

```
EXIT
```

command. You will be asked to confirm that you wish to continue with the close-down, and the system will perform a brief wrap-up operation and then return you to the:

```
C:>
```

or similar DOS prompt. At this point, you may then use DOS in the normal manner or you may switch off your machine.

The various routes through the system are illustrated at the end of this *MB-Guide.*

The keyboard

Within Advanced Pick, the keyboard is given extended functionality. The new actions are available whenever you are using:

* The TCL stacker. You will use the extended keyboard facilities to browse through the stack and to amend the contents. We shall talk about the AP TCL stacker in a later article.

* The Update Processor. You will use the extended keyboard to create, change, maintain and/or delete the items and the data fields on your files.

This means that many of the functions of the TCL stacker and the Update Processor are invoked by typing simple key sequences, rather than explicit commands (such as are used by the Pick editor). Typically, these sequences consist of the <Ctrl> key followed by one or more letters.

The keyboard sequences allow you to move through the stack or through a data item and to perform a large number of other actions. For example, the sequence:

<Ctrl> K

or, to use the underlined notation adopted in the Advanced Pick Reference Manual:

K
will move the cursor one place to the right; the sequence:

<Ctrl> J or J
will move the cursor one place to the left; the sequence:

<Ctrl> L or L
will delete a single character; the sequence:

<Ctrl> R or R
will enable/disable the insertion of text. Finally, the sequence:

<Ctrl> X F or XF
will file the item; the sequence:

<Ctrl> X O or XO
will delete the item, and so on.

There are a great many such actions. Some of the more frequently-used are summarised at the end of this *MB-Guide*.

Another important feature of general keyboard usage - notably TCL - is that almost all input is case *insensitive*. Thus, a TCL command may be entered as either:

LIST STOCK
or

```
list stock
```

with the same effect. Within a Basic program, the CASING ON/OFF statement can be used to specify whether keyboard input data is to be case sensitive or not.

When using the Update Processor, there is no limit to the length of an input line, but on AP/DOS, the type-ahead buffer is only 15 characters long.

Under AP/DOS, the full scope of the keyboard may be available, including:

* PRINT SCREEN - to print a copy of the current screen.
* INSERT
* PAGE UP
* PAGE DOWN
* DELETE

although the normal cursor control keys (left-arrow, right-arrow, up-arrow, down-arrow) cannot be used with the Update Processor unless you have created and reset a KEYBOARDS definition to set-up your keyboard correctly.

7 TCL and TCL commands

TCL is the primitive operating medium for Advanced Pick, as with all previous versions of the Pick system.

The following general points apply:

* The TCL prompt has been changed from the > of R83 to the:

```
:
```

character.

* A null TCL command will display the time and date immediately before the TCL prompt. This can be enabled/disabled by the commands:

```
TIMEDATE-ON
TIMEDATE-OFF
```

* There are a number of commands, such as TIMEDATE-ON and TIMEDATE-OFF, which enable/disable specific functions. In general, these commands are explicit of the form:

```
TIMEDATE-ON
TIMEDATE-OFF
```

or they may be entered as:

```
TIMEDATE (N
TIMEDATE (F
```

respectively. The simple command:

```
TIMEDATE
```

with no options will display the current status of the function. Other examples include:

```
BREAK-KEY-OFF / BREAK-KEY-ON
CASE-OFF / CASE-ON
LEGEND-OFF / LEGEND-ON
SPELLER-OFF / SPELLER-ON
STACK-OFF / STACK-ON
TCL-HDR-OFF / TCL-HDR-ON
TYPE-AHEAD-OFF / TYPE-AHEAD-ON
```

* Any TCL command may be entered in either UPPER- or lower-case. Thus,

```
sort stock
SORT STOCK
sort STOCK
SoRt StOcK
Sort Stock
```

and so on, are all equivalent. This is a potential mine-field and great care should be taken when choosing filenames and item-ids.

* A utility called R83.SETUP can be used to invoke the set of commands:

BRK-DEBUG	the <Break> key will invoke the debugger;
ESC-DATA	the <Esc> key is a normal data key;
SPELLER-OFF	disable the spelling checker;
LEGEND-OFF	disable the pick legend on Spooler output;
TCL-HDR-OFF	disable the TCL header on Spooler output;
TIME-DATE-OFF	disable the time/date display at TCL;
WHO (C	set the WHO output as described below;
TERM ,,,,2	set the form-feed delay of the terminal characteristics to 2;

  which, when used in conjunction with the <Caps Lock> key will cause the system to behave like a standard R83 system.

* There are duplicate definitions for many verbs and keywords both with and without hyphens. Thus, CREATE-FILE may be entered as CREATEFILE. This is made possible by having MD entries for both CREATE-FILE and CREATEFILE. However, the alternate entries are not universally available, so it is probably better to stay with the standard and hyphenated forms.

* There are also abbreviated forms for many verbs and keywords. Thus, CREATE-FILE may be entered as CF. This is made possible by having MD entries for both CREATE-FILE and CF, but such abbreviations are not universally available, so it is probably better to stay with the full forms.

* The format of the MD entries for the TCL verbs has changed.

* Many TCL activities are now performed by Basic programs. This means that they are *get-at-able* and can be inspected to see the true action of the activity, and - in extreme circumstances - they may even be modified for your own requirements.

* The TCL input buffer was 140 characters in length on R83. On Advanced Pick, the input buffer is virtually unlimited in size.

* The nature and output of many familiar commands has been modified. In particular, where R83 systems used the account-name to identify the current user, AP uses the user-id in some contexts and the master-dictionary name in others.

* The on-line HELP facility provided by the EPICK (encyclopaedia Pick) can be called up by commands of the form:

```
HELP xxxxx
```

There are one or two specific points:

* Certain TCL commands assume the * if an item-list is omitted:

```
EDIT STOCK
```

```
COPY STOCK
CT STOCK
```

in contrast to that of R83 which would require an item-list to be specified.

Note that the Update Processor requires the * to process all items, otherwise it will generate a new random item-id and create a new item.

* Certain Access sentences will now accept item-ids with or without quotes:

```
T-DUMP STOCK 1234 3456
```

* The TCL stacker is standard on Advanced Pick and will store an unlimited number of TCL commands for future use. We discuss the stacker in a separate section.

* All the TCL commands issued by the user may be recorded on a file for inspection. The commands are recorded on the file CAPTCL and the lists of commands are held with item-ids of the form:

```
u*p
```

where u is the user-id and p is the port-number.

The recording is enabled/disabled by the:

```
CAPTURE-ON
CAPTURE-OFF
```

commands or the equivalent:

```
CAPT (N
CAPT (F
```

commands. This is quite distinct from the stacks which are stored by the TCL stacker.

* The word DATA is optional on the CLEAR-FILE command.

* The POVF command (and the synonyms OVERFLOW and OVF) outputs a summary of the space available. There are options: A to output the FIDs and numbers of frames (this reproduces the normal R83 report); B to output just the numbers of frames available; n-m to include only those frames with FIDs in the specified range.

* The TANDEM facility is available to link the current process to any other process, so that input at either port affects both.

* The WHO command displays the port-number (p), the user-id (u) and the master dictionary name (d) in the form:

```
p u d
```

The command:

```
WHO (C
```

will specify that the WHO command (and the equivalent U50BB user-exit) is to display the information in the form:

```
p d u
```

as some application systems expect the master dictionary (account) name to be the second field. This format is associated with the user-id and is retained over LOGTO and TO commands and until the user logs off.

* A number of new TCL commands have been introduced for use with macros and menus: COMMENT, DISPLAY and PROMPT.

A large number of additional TCL commands are available. We list these below. Those marked *L are associated with level-pushing; *P are associated with the phantom processor; *T are associated with the transaction logger. Full details can be found in the Advanced Pick Reference Manual.

ABS.FID	EXPORT	RESTORE-ACCOUNTS
ADD	F-RESIZE	RMBI
ADDBI	FID	RUN-LIST
ASSIGNFQ	FRAME-FAULT	SEARCH
BOOTSTRAP	IMPORT	SEARCH-SYSTEM
BRK-DEBUG *L	INIT-OVF	SEND-MESSAGE / SM
BRK-LEVEL *L	ISELECT	SET-HALF
CAPT	ISSELECT	SET-OVF-OVF
CAPTURE-OFF	LD	SET-RUNAWAY-LIMIT
CAPTURE-ON	LDF	SH
CASE	LEGEND	SPELLER
CHECK-WS	LEGEND-OFF	STACK-OFF
CHECK.DX	LEGEND-ON	STACK-ON
CHKSUM	LIST-COMMANDS	STARTLOG *T
COLDSTART	LIST-FILES	STARTSCHED *P
COLDSTART.LOG	LIST-JOBS	STEAL-FILE
COMMENT	LIST-LOGOFFS	STOPLOG *T
COMPILE	LIST-MACROS	STOPSCHED *P
COMPILE-CATALOG	LIST-MENU	SUB
COMPILE-RUN	LIST-MENUS	TCL *L
CONV-CASE	LIST-PIBS	TCL-HDR
CP	LIST-RESTORE-ERRORS	TCLOG
CREATE-ABS	LIST-USERS	TERMP
CREATE-INDEX	LIST-VERBS	TIMEDATE
CREATE-MACRO	LISTBI	TOTAL-ON
DATE	LISTC	TXLOG-STATUS *T
DEBUG	LOG-MSG	U / UP / UPDATE
DELETE-INDEX	MD-RESTORE	UD
DIAG	MUL	UNLOCK-GROUP
DISC	NFRAME-INDEX	UPDATE-ACCOUNTS
DISPLAY	NSELECT	UPDATE-MD
DIV	OP	VERIFY-ABS
END	OVERFLOW / OVF	VERIFY-INDEX
ESC-DATA *L	PROMPT	WHICH
ESC-LEVEL *L	REBUILD-OVF	Z{H}{S} *P
ESC-TOGGLE *L	RECOVER-ITEM / RI	
EXEC	RENAME-FILE	

7.1 Stacker

The Advanced Pick TCL stacker allows you to recall a TCL command which you entered earlier and issue it again.

The stacker is normally active, but it may switched off/on by means of the STACK-OFF and STACK-ON commands.

When the stacker is active, all TCL input is automatically added to the stack as are those commands invoked through macros and menus (except those *macros* with the N identifier). If you re-issue a TCL command which you issued previously, the original command will be moved at the top of the stack; this avoids your filling the stack with copies of identical commands.

The contents of the stack can be inspected by the standard Update Processor facilities, allowing you to:

* To search for a specific substring within the stack. We describe this is a separate section.

* To navigate up and down the stack by means of the <Ctrl> D and <Ctrl> F sequences.

* To re-issue a command by pressing <Return> when the required command is reached.

* To amend the displayed command by using the cursor movement and editing sequences:

  <Ctrl> J cursor left one character
  <Ctrl> K cursor right one character
  <Ctrl> L delete a character
  <Ctrl> Y cursor left one word
  <Ctrl> U cursor right one word
  <Ctrl> O delete a word
  <Ctrl> W insert a blank space
  <Ctrl> R enable/disable text insertion

* To delete a command by means of the <Ctrl> E sequence.

* To leave the inspection of the stack by means of the <Ctrl> X sequence.

The stacks of TCL commands are held on the TCL-STACK file which is held on the DM account, each separate user's list being held with the user-id as the item-id.

There is no limit to the number of commands which are held in the stack, so you must take care to clear it down (or delete the item) from time to time by means of a sequence such as:

```
     UPDATE TCL-STACK,username
or
     UPDATE DM,TCL-STACK, username
```

7.2 DOS commands at TCL

There are a number of new TCL commands which are provided specifically for use with DOS:

ADDBI		DISC		LISTBI		RENAME	[D]
ATTRIB	[D]	DISKCOMP	[D]	MKDIR	[D]	RESTORE	[D]
BACKUP	[D]	DISKCOPY	[D]	MODE COM	[D]	RMBI	
CHDIR	[D]	EDLIN	[D]	MODE LPT	[D]	RMDIR	[D]
CHKDKS	[D]	ERASE	[D]	MODE	[D]	SET	[D]
CLS		EXIT		MORE	[D]	TREE	[D]
COMP	[D]	EXPORT		PATH	[D]	TYPE	[D]
DEL	[D]	FORMAT	[D]	PC-COPY	[D]	XCOPY	[D]
DIR	[D]	IMPORT		PROMPT	[D]		

Those marked with [D] invoke the equivalent DOS command.

If your configuration is suitable, you may also invoke DOS commands directly from TCL by issuing commands of the form:

```
!DIR
```

prefixing the DOS command by the ! character, and from within Basic programs by statements of the form:

```
EXECUTE '!DIR'
```

Macros

Advanced Pick offers the concept of *macros.* A macro allows you to establish an item containing a sequence of TCL commands which can be invoked by a single command. The purpose of a macro is similar to that of a Proc, although the construction is much simpler.

The general format of a macro-definition item is:

```
000 macro name
001 M or N {comments}
002 TCL command 1]data 1]data 2] ...
003 TCL command 2]data 1]data 2] ...
004 TCL command 3]data 1]data 2] ...
:
n   TCL command m]data 1]data 2]
```

The first attribute of the macro-definition item starts with the identifier M or N followed by optional comments. If M is used, then - as each TCL command in the macro is executed - it is added to the stack of TCL commands and passed to the user to modify/issue. If N is used, the commands of the macro are executed without any such user-intervention.

Each attribute from attribute 2 onwards holds a set of values, representing:

1) The TCL command which is to be executed, followed by

2) An optional stack of input data which is to feed the TCL command.

The macro processor will handle any number of TCL commands.

A typical example might be:

```
000 STOCK.MACRO.1
001 M
002 SORT STOCK BY PART.NO LPTR
```

or, if the macro is to invoke several commands one after another:

```
000 STOCK.MACRO.2
001 M THE MAIN STOCK CONTROL ROUTINES FOR THE STORES
002 SORT STOCK BY PART.NO LPTR
003 COPY STOCK * (O)](STOCK.BACKUP
004 RUN PROGS STOCK.MAINT
005 CLEAR-FILE DATA STOCK.BACKUP
```

The macro is invoked by typing the name of the macro - STOCK.MACRO.1 or STOCK.MACRO.2 as if it were a TCL command. Thus, when this latter macro is invoked by means of the TCL

command:

```
STOCK.MACRO.2
```

the macro processor will display each of the TCL commands in turn and wait for you to hit <Return> to execute the command. Before you hit <Return>, you may use the standard cursor controls of the Update Processor to amend any part of the display to patch the TCL command before it is invoked. The <Ctrl>X command will ignore the command.

If any of the TCL commands (or the patched TCL commands) is invalid, then the macro will abandon.

All macro-definition items are held on the MD and may be created by means of the Update Processor:

```
UPDATE MD macro.name
```

or by means of the CREATE-MACRO command:

```
CREATE-MACRO macro.name
```

or

```
CREATE-MACRO macro.name (N
```

which will establish *the last executed TCL command* (taken from the top of the TCL stack) and save it on the MD as a macro with the name *macro.name*. The N option on the CREATE-MACRO command will establish the macro with an N identifier in attribute 1.

Currently, it is not possible to hold macros on any file other than the MD. The command:

```
LIST-MACROS filename
```

will list the macros on the DICT section of the specified file.

If a macros is invoked by a command such as:

```
macro.name text
```

then the specified *text* will be appended to the end of the first command in the macro. Apart from this facility, it is not possible to tailor the contents of a macro to incorporate data supplied at execution time.

A number of TCL commands are provided specifically for use in macros: these are COMMENT, DISPLAY and PROMPT. They are all used to display information on the screen.

The COMMENT command is used to display output on the screen and produce special effects and cursor-positioning and has the form:

```
COMMENT {text} {text} ... {text} {+}
```

where *text* is an expression of the following form:

(c,r) to specify the column (c) and row (r) position for the cursor. This is similar to the @(c,r) function in Basic and the (c,r) element in the Proc T statement.

(-n) to produce one of the cursor control effects. This is similar to the (-n) element in the Proc T statement and the @(-n) function in Basic.

(xn) to output the character whose decimal value is n. This is similar to the In element in the Proc T statement.

(xxxxx) to output the string xxxxx.

+ to hold the cursor at the end of the output string.

The DISPLAY command has the form:

```
DISPLAY message
```

and will display the message on the screen.

The PROMPT command has the form:

```
PROMPT {message}
```

and will display the message:

```
{message} -- Quit/Continue (q/c) ?
```

on the screen and accept a response of Q or C from the user. A response of Q will abort the macro/menu, a response of C will continue with the macro/menu.

8.1 Menus

Advanced Pick offers the concept of *menus*, allowing you to set up a system of TCL commands which can be invoked by menu selection.

The general format of a menu-definition item is:

```
000 menu name
001 ME
002 menu title
003 descriptive text 1]Help message 1]TCL command 1
004 descriptive text 2]Help message 2]TCL command 2
005 descriptive text 3]Help message 3]TCL command 3
:
n   descriptive text m]Help message m]TCL command m
```

The first attribute of the menu-definition item contains the identifier:

```
ME
```

followed by optional comments on the same line.

The second attribute contains the title which is to be displayed at the head of the menu screen. This heading may include the codes:

'i' to display the name of the menu,

'd' to display the date,

't' to display the time and the date,

'p' to display the page number.

Note that these codes must be followed by a space.

For each option to be offered on the menu, attributes 3 onwards will hold a set of three values, representing:

1) A *descriptive text* which is to be shown alongside that option on the menu.

2) An optional *help message* which is to be displayed if the user solicits further information about that option.

3) The *TCL command* which that option is to invoke. This may be any TCL command and even another menu name.

   Several TCL commands may be specified as a sequence of *multivalues*.

A typical example might be:

```
000 STOCK.MENU
001 ME The main stock control menu for the stores
002 Stock Control Processing
003 Display stock items
    Display a list of the records on the STOCK file
    SORT STOCK BY PART.NO
004 Print stock items
    Produce a printed list of the records on the STOCK file
    SORT STOCK BY PART.NO LPTR
005 Add new stock records
    Add new records to the STOCK file
    RUN STOCK.BP ADD.REC
006 Change stock records
    Change existing records on the STOCK file
    RUN STOCK.BP MOD.REC
007 Delete stock records
    Delete old records from the STOCK file
    RUN STOCK.BP DEL.REC
```

When this menu-definition item is invoked by means of the command:

```
STOCK.MENU
```

the menu processor will interpret the menu-definition item and:

* Display the menu title
* Display the available options and an associated numeric control code which will invoke that option
* Display an input prompt message
* Accept the user's control code and react accordingly

In this instance, the screen image will be:

```
                    Stock Control Processing

1) Display stock items              4) Change stock records
2) Print stock items                5) Delete stock records
3) Add new stock records

Enter number of choice, number ? for Help, <Return> to exit menu,
or verb:
```

The user may then enter the required option:

* A response of, say, 2 will select option 2 (the Print stock items option) and invoke the TCL command:

  ```
  SORT STOCK BY PART.NO (PTR
  ```

* A response of, say, ?2 or 2? will display the appropriate help message:

  ```
  Produce a printed list of the stock items
  ```

* A response of <Return> will drop out of the menu, and pass to TCL.

* Any other response will be regarded as a TCL command and executed.

When the appropriate action has been taken, the user will be asked to:

```
Hit any key to return to the Menu
```

and the menu will be redisplayed and the action repeated.

The menu processor will handle any number of options, and the screen display will be tailored to handle varying lengths of descriptive text, producing only a single list if the texts are long.

Menu-definition items are held on the MD and may be created by means of the Update Processor:

```
UPDATE MD menuname
```

Currently, it is not possible to hold menus on any file other than the MD. There are two associated commands:

```
LIST-MENU filename {itemlist}
```

which will produce a formatted listing of the menu(s) on the specified file.

```
LIST-MENUS filename
```

which will list all the menus on the DICT section of the specified file. Since all menus are normally held on the MD, the practical form of these commands are:

```
LIST-MENU MD {itemlist}
```

and:

```
LIST-MENUS MD
```

If a macros is invoked by a command such as:

```
macroname text
```

then the specified *text* will be appended to the end of the first command in the macro.

Level pushing

When you are carrying out any processing - be it a long Access report, an update program, or whatever - your terminal will be locked out until the process has finished and control returns to you. This can be inconvenient.

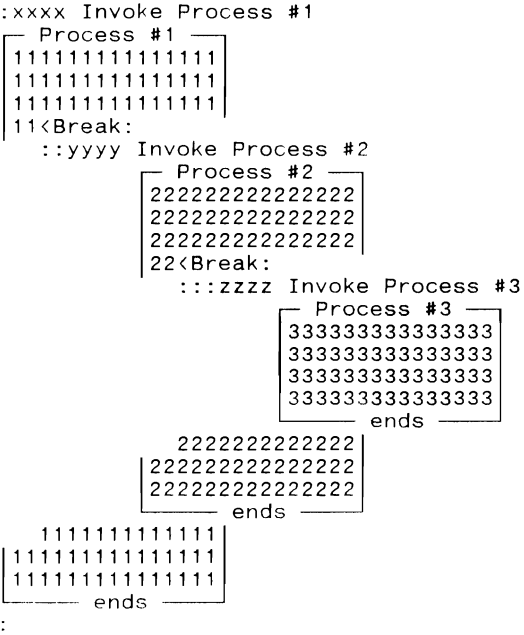
When you are in the midst of a process, Advanced Pick allows you to press the <Esc> key to suspend the current process and then issue a TCL command. When you interrupt a process in this manner, the system stops execution and saves all the necessary parameters so that the execution can be resumed exactly where it was interrupted, it then replaces the normal system prompt by:

```
::
```

to indicate the level at which you are processing. You may then issue any TCL command to initiate a second process. The second process may, in turn, be interrupted and a further operation invoked. When each process has completed at the lower level, the appropriate prompt:

```
::
```

will be displayed, and you may issue a further TCL command at this lower level.



The following special TCL responses are available when the:

```
:::
```

prompt is displayed:

<Return>	to return to the previous level and continue execution of the process there.
END	to terminate the process at the *lower* level. This action makes the level-pushing feature similar to the action of the <Break> key under R83.
OFF	to log the user off the system and terminate all processes.
DEBUG	to enter the interactive system debugger, or, if the execution of a Basic program has been interrupted, to enter the Basic symbolic debugger.

To avoid confusing users who are familiar with the normal action of the <Break> key, the <Esc> key may be used instead of the <Break> key for level-pushing. The following TCL commands are available to control this:

BRK-LEVEL
will assign the <Break> for level-pushing,

ESC-LEVEL
will assign the <Esc> for level-pushing,

BRK-DEBUG
will assign the <Break> for invoking the debugger (the conventional role),

ESC-DATA
will assign the <Esc> for sending the <ESCAPE> character (the conventional role).

ESC-TOGGLE
will switch the action of the <Esc> key from level-pushing to sending the <ESCAPE> character, and vice versa.

If the ESC-LEVEL is enabled, you can still generate the <Esc> data character by means of the sequence:

<Ctrl> [

The depth of the processing level for each user is now shown in the STAT field of the modified WHERE report:

```
Ln  PCB    PIB   ABS    Stat R1 & Return stack contents
    FID    Stat  Base

000 00215A FF10 000012    6 ws.where1:01E ws.who:236
                            |
          depth of nesting --
```

The processing may be interrupted and nested up to 16

levels, beyond this depth, any interruptions will be ignored.

10 Phantom processes

This command invokes a phantom process and then allows the user to continue with other work whilst the phantom process is carried out in the background.

The general format of the command is:

```
Z{H}{S} tclcommand
```

The H option will send all the terminal output to a spooler hold file, and the S option will suppress any error messages which may be generated generated by the phantom task. Without the S option, the phantom process will display all messages on the terminal at which the task was initiated.

If the Phantom Processor is invoked by the simple commands Z or ZH or ZHS without specifying a TCL command, then the user will be asked for:

1) The user-id (and the password) which is to be used. The default is the current user-id.

2) The name of the account (and the password) which is to be used. The default is the current master dictionary.

3) The TCL command which is to be invoked,

4) The line on which the process is to be run. The default is any available phantom line.

5) Any input data to be submitted to the command. A blank data line is represented by <Ctrl> N, and the end of the input data is indicated by a null line.

The phantom task will then be initiated and the user's terminal will be freed for normal use.

The JOBS file holds details of all jobs handled by the phantom processor.

The TCL commands STARTSCHED and STOPSCHED are used to enable/disable the phantom processor.

11 Transaction logging

Transaction logging is a feature in which any changes made to a file are logged immediately to the backing storage device. In the event of a system failure, the system may be restored from the last file-save tape and then the logged transactions rolled forward to recover the system to the state at the time of the failure.

Only files whose D-pointer items have an L in attribute 1 are involved in transaction logging. This can be set by specifying the L option on the CREATE-FILE command which creates the file, and may be modified by means of the Update Processor.

The TCL commands STARTLOG and STOPLOG are used to enable/disable the transaction logger. Any transaction logging which is carried out whilst the logger is suspended are written to disk and dumped to backing storage when the logger is restarted. The TCL command TXLOG-STATUS will display the transaction logger status

12 Update Processor

The Update Processor performs a number of rôles:

* As a general purpose editor, replacing the EDIT verb. The standard Editor is still available in *Advanced Pick.*

* As an application tool.

But, in general, the complexity of the key sequences which are used to control the processor (and the fact that these cannot generally be re-assigned for a specific application) mean that the Update Processor is really a tool for the *technical* user - the programmer - rather than for the *non-technical* end-user. The end-user will be supported by applications software.

The Update Processor is invoked by a command of the form:

```
     UPDATE filename {itemlist} {(options}
or
     UP filename {itemlist} {(options}
or
     U filename {itemlist} {(options}
```

where *filename* is the file which is to be maintained, and *itemlist* is the list of item-ids which are to be processed.

The options are:

C to clear the screen before each item is presented;

I to bring the item-id into the UP workspace where it can be amended just like any other attribute. By changing the item-id, you can leave the current item and move to another item on the file. This is identical to the action of the ID-PROMPT modifier.

L to indicate that the processing is to be Look only and items cannot be updated/filed;

S to disable the spelling checker on entry to the UP.

There are several special forms:

UPDATE STOCK
: will create a unique item-id (based upon the current date and a system-wide sequence number) and then proceed to accept the data for that item. When the item is filed (by the <Ctrl> X F sequence), control will return to TCL.

UPDATE STOCK DATE-ENTRY
: will create a unique item-id (based upon the current date and a system-wide sequence number) and then proceed to accept the data for that item. As each item is filed (by the <Ctrl> X F sequence), a new item will be presented. The sequence <Ctrl> X K will terminate the activity and return to TCL.

UPDATE STOCK item1 item2
will apply the Update Processor to the specified items. When one item is filed (by the <Ctrl> X F sequence), the next item will be presented.

The list of items can be abandoned by means of the <Ctrl> X K sequence.

The sequence <Ctrl> X B will leave the current item and return to the previous item in the list.

As the item is being modified, the <Ctrl> keys are used to move through and modify the item.

In all cases, if there is a macro held in attribute 15 of the data-level identifier for the file, the attributes shown there will be displayed as field identifiers before the data is accepted for the item(s). If there is no macro list, then the line-numbers 001, 002, and so on, will be displayed for each input line (as with the standard EDITor).

The special form:

UD filename itemlist {(options}

is used for editing items - such as attribute definitions - which are held on the dictionary section of the file. D-pointers cannot be edited by means of the standard Editor.

The Update Processor has facilities specifically for use with Basic program items:

* The sequence <Ctrl> X R will file, compile and run the item,

* The sequence <Ctrl> X C will file, compile and catalogue the item.

12.1 Bridges

The concept of *referential integrity* between the data on our files demands that, if the CLIENT record for client 1234 indicates that this client submitted order-number 234, then the ORDER record 234 must also indicate that this was submitted by client number 1201.

```
    CLIENT
000 1201                    <== client number
001 JONES BROS
002 34 HIGH STREET
003 234]345]567             <== Order numbers
```

```
    ORDER
000 234                     <== Order number
001 1201                    <== client number
002 8166
003 25000
```

Most database management systems require considerable programming effort to enforce referential integrity between the files. Advanced Pick controls the referential integrity of the files automatically by the concept of *bridging.*

There is an implied (logical) bridge between the order-number in attribute 3 of the CLIENT file and attribute 0 of the ORDER file, and (in the opposite direction) between the client-number in attribute 1 of the ORDER file and attribute 0 of the CLIENT file: for each client number in the ORDER items, there must be a corresponding client on the CLIENT file, and for each order number in the CLIENT items, there must be a corresponding order on the ORDER file.

The B processing code will allow us to specify such a relationship between the files. We would use the correlative:

```
BCLIENT;1;3
```

in the data-level identifier item of the ORDER file to establish such a bridge to operate in both directions, from the ORDER file to the CLIENT and from the CLIENT file to the ORDER.

Whenever an item is added to or deleted from the ORDER file, attribute 3 of the CLIENT file is changed accordingly. Furthermore, in this situation, the Update Processor will not allow the order-numbers on the CLIENT file to be deleted until the corresponding order on the ORDER file has been deleted. We discuss the B processing code in a separate section.

12.2 Cruising

If an index is available for an attribute which is being handled by the Update Processor, the user may cruise through this index looking for the required value. To illustrate this, let us imagine that an employee's surname is to be entered and that an index is being maintained for the surname. If the user enters a name or part of a name, such as:

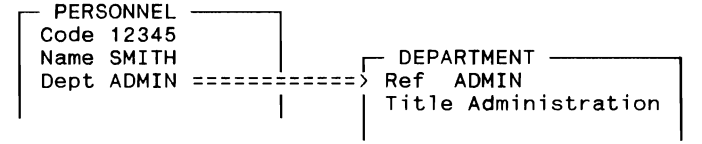
```
S
```

followed by the <Ctrl> U key, all the surnames starting with S will be presented one at a time as the <Ctrl> U key is repeatedly pressed. The <Ctrl> Y key can be used to go back up the index. When the required name is presented, the user will press <Return> and that name will be placed into the item which is being updated.

12.3 Zooming

Whilst using the Update Processor, the concept of zooming allows the user to take an attribute in the current item and use this as a key to go to a second file and processor the corresponding item there.

Let us imagine that you are modifying an item on the PERSONNEL file and that the cursor is positioned at the DEPT field where the current value is ADMIN:



If we press the sequence:

<Ctrl> G G

the Update Processor will jump across to allow us to modify (or create) the item on the DEPARTMENT file with the item-id ADMIN. When we use a <Ctrl> F sequence to terminate the activity on the DEPARTMENT file, we shall return to the PERSONNEL item.

If, whilst the cursor is at the DEPT field of the PERSONNEL item, we press the keys:

<Ctrl> Y

or

<Ctrl> U

the Update Processor will cruise through the index of the DEPARTMENT file displaying the TITLEs for each item. When we press <Return> the appropriate REF for that DEPARTMENT item will be placed into the DEPT field of the current PERSONNEL item. Thus, if you enter a value (or a part of a value) for the DEPT:

ADMIN

and then press the <Ctrl> Y key, you will cruise backwards through the index of DEPARTMENTs starting at ADMIN, and the <Ctrl> U key will cruise forwards through the DEPARTMENT index. When the required name is presented, the user will press <Return> and that REF will be placed into the PERSONNEL item which is being updated.

When the DEPT is entered into the PERSONNEL item, the Update Processor will display the full TITLE of that DEPARTMENT.

These actions require that:

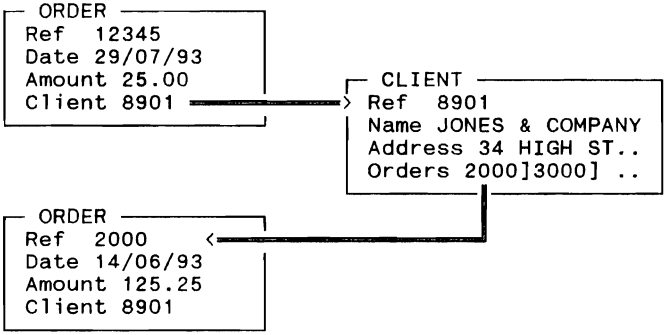
* There is an index for the REF field of the DEPARTMENT file,

* There is an I index correlative in the DEPT field of the PERSONNEL file which refers to the index for the DEPARTMENT file.

* There is a Tfile correlative in the DEPT field of the PERSONNEL file,

12.4 Double-clutching

Let us imagine that we are using the Update Processor to modify item 12345 on the ORDER file, as in the diagram below. The concept of *double-clutching* combines the actions of zooming and cruising to allow us to use the current CLIENT field 8901 to pass to that item on the CLIENT file, and then to cruise through the items on the ORDER file for that client.



The sequence <Ctrl> G G is used to make both transfers. The zooming to another file may continue virtually indefinitely.

At each stage, you may make any changes to the item which is currently active, leaving (and returning to the previously-active item) by means of the <Ctrl> X F or <Ctrl> X X sequence.

These actions require that there are indexes for the CLIENT and the ORDER file.

Care must be taken that you do not attempt to change an item which you (or indeed any other user) is already editing. If this happens, the system will report:

```
'xxx' locked by line yy
retry/look/quit (r/l/q)?
```

12.5 Prestored commands

As with the R83 Pick EDITor, the Update Processor has facility to create and execute a prestore command or sequence of commands. The prestore command(s) are held in a single prestore buffer, and a number of commands are available for handling this buffer.

1) Create and modify the prestore buffer by the sequence:

<Ctrl> Z L

This invokes the Update Processor to modify the contents of the prestore buffer. Commands are entered entered into the buffer without the <Ctrl> key and text operands are enclosed in ' or " characters.

2) Process the contents of the prestore buffer by the sequence:

    <Ctrl> P

3) Save the current contents of the prestore buffer as an item on a file by the sequence:

    <Ctrl> Z W

(that is, prestore Write.) and when the Update Processor asks for the prestore id, enter:

    prestore.id <Return>

specifying the id by which the prestore section is to be saved.

4) Load a previously saved item into the prestore buffer by the sequence:

    <Ctrl> Z R

(that is, prestore Read), and when the Update Processor asks for the prestore id, enter:

    prestore.id <Return>

specifying the id by which the prestore section was previously saved.

5) Load and then execute a previously saved item into the prestore buffer by the sequence:

    <Ctrl> Z R

(that is, prestore Read), and when the Update Processor asks for the prestore id, enter:

    prestore.id <Ctrl> P

specifying the id by which the prestore section was previously saved.

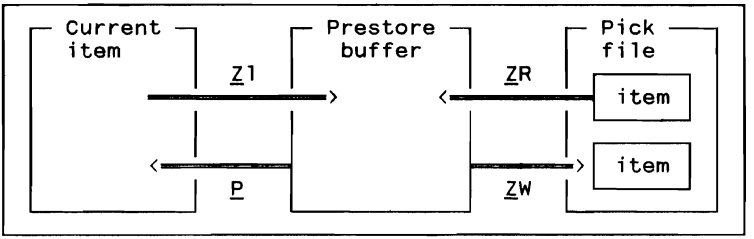
If the prestore id is entered in the form:

    (filename itemid

the prestore sections can be held as items on a specific file. If only an id is specified, then it is assumed that the prestore commands are held on the MD.

The format of the prestore item is:

```
000 prestore.id
001  P
002  xxxxxxxxxxx      <== Update processor commands
```



12.6 Cutting and pasting

The Update Processor has facilities for handling a block of text using the technique known as *cut and paste.*

At any one time, there is a single *cut buffer* available for holding text which has been cut and/or is to be pasted into an item. The following actions are available for handling the cut buffer. They all start with <Ctrl> C (for cutting).

1) Mark and delete a block of text by the sequence:

<Ctrl> C D

to mark the start of the block, and

<Ctrl> C C

to mark the end of the block. The cut block is deleted from the document and loaded into the cut buffer.

2) Mark a block of text and place it in the cut buffer by the sequence:

<Ctrl> C L

to mark the start of the block, and

<Ctrl> C C

to mark the end of the block. The document is not changed, but the marked block is loaded into the cut buffer.

3) Paste in a block of text: The cursor is first moved to the required position, and the current contents of the cut buffer are pasted into position by the sequence:

<Ctrl> C P

You should remember that the cut buffer is initialsed each time you invoke the Update Processor, so it is not possible to *carry* text from one item to another (except by writing/reading the cut buffer, as described below).

4) Save the current contents of the cut buffer as an item on a file by the sequence:

<Ctrl> C W

(that is, Cut Write) and then, when the Update Processor asks for the cut id, enter:

```
cut.id <Return>
```

specifying the id by which the cut section is to be saved.

5) Paste a previously cut section by the sequence: the cursor is first moved to the required position.

```
<Ctrl> C R
```

(that is, Cut Read), and when the Update Processor asks for the cut id, enter:

```
cut.id <Return>
```

specifying the id by which the cut section was previously saved. This action is similar to the ME command of the Pick EDITor.

If the cut id is entered in the form:

```
(filename itemid
```

the cut sections can be held as items on a specific file. If only an id is specified, the cuts are held on the MD.

The cut buffer is empty when the Update Processor is invoked, and the contents are lost when the UP session terminates; thus, the buffer can be used to pass text from one item to another in the same UP session, but not between sessions.

12.7 Searching

The Update Processor offers search and/or replace facilities. This is accomplished by a number of different sequences:

1) To search only

The sequence:

```
<Ctrl> A string <Return>
```

or

```
<Ctrl> A string <Ctrl> M
```

will search for the specified *string*.

2) To search and replace

The sequence:

```
<Ctrl> A string1 <Ctrl> R
```

and when the system asks for a replacement string:

```
string 2 <Return>
```

will search for and replace occurrences of *string1* by *string2*. When an occurrence is found, the system will accept a further response from the user. This will be any of:

<Ctrl> A
to ignore this occurrence and find the next,

<Ctrl> N
to replace this and all further occurrences,

<Ctrl> R
to replace this occurrence and find the next,

<Ctrl> X
abandon the search/replace.

3) To replace all occurrences

The sequence:

<Ctrl> A string1 <Ctrl> R

and when the system asks for a replacement string:

string 2 <Ctrl> N

will replace all occurrences of *string1* by *string2*, without any intervention from the user.

4) To repeat the search

The sequence:

<Ctrl> A

at any point, will repeat the last search,

These key sequences are summarised at the end of this *MB-Guide*.

12.8 Spelling check

The Update Processor has an automatic spelling checker based upon the WORDS file held on the DM account. Whenever an unrecognised word is encountered, the processor will beep and allow the user to enter any of:

<Ctrl> A
to accept the word.

<Ctrl> A <Ctrl> A
to accept the word and, when it is complete, add it to the WORDS file.

<Ctrl> A <Ctrl> A <Ctrl> A
to disable the spelling checker.

<Ctrl> U
to scan the WORDS file and find the next acceptable

word.

<Ctrl> Y
to scan the WORDS file and find the previous acceptable word.

<Ctrl> Z S
to switch the spelling checker on/off. If the Update Processor is invoked with the S option, the spelling checker will be disabled.

The TCL commands SPELLER/SPELLER-ON/SPELLER-OFF are used to interrogate/enable/disable the spelling checker.

Before these facilities can be used, an index for the WORDS file must be created by means of the command:

```
CREATE-INDEX WORDS A0
```

13 Output Processor

The Output Processor - OP - is essentially an updated version of Runoff and is used to produce a document according to a set of text lines and OP commands held in an item.

Although a great many OP features and commands are identical to those of Runoff (and Runoff is still available for compatibility), those users who are familiar with Runoff will need to know the differences between this and OP:

* Output Processor commands may be embedded anywhere within the text. It is not necessary to hold them on separate command lines, as is the case with Runoff commands.

* All active commands, such as .JUSTIFY, are disabled by the X-form .XJUSTIFY.

* The concept of Output Processor macros allows the user to set up standard items which can be called in much the same way as with .READ commands. These macros are held on a file and have item-ids of the form

  .xxxxx

  and the macros are invoked simply by including the item-id within the text, just like a normal Output Processor command. The name of the file which holds the macros is declared on a .MACRO FILE command. If no such command has been used, the current file is assumed to be the macro file.

* There are facilities for automatic numbering of figures and tables. The number is of the form:

  ccc-nnn

  where *ccc* is the chapter number and *nnn* is the sequential number for the figure or the table.

* There is a greater recognition of the facilities of current printers, with facilities for subscripts, superscripts and fonts.

* Although Runoff and the Output Processor are similar, the two are not compatible and Runoff documents cannot be handled by the Output Processor.

The Advanced Pick Reference Manual gives full details of the OP commands. The following are noteworthy for the reader who has used Runoff.

.appendix / .apx	.set chapter / .sch
.block center / .bc	.subscript / .sub
.boldface / .bf	.superscript / .sup
.char	.tab left / .tl
.col	.tab right / .tr
.columns set / .columns	.tab rightm / .trm
.cursor	.table / .tbl
.date	.tc heading / .tch

.default / .df	.tcl
.em	.tcl box / .tclbox
.figure / .fig	.underline / .ul
.font / .f	.underline words / .uw
.gohanging tab / .ght	.variable columns / .vcol
.hanging tab / .ht	.vmi
.hyphenate / .hy	.x
.indent rmargin / .irm	.xblock centre / .xbc
.index heading / .ixh	.xboldface / .xbf
.italics / .it	.xcol
.lpi	.xcolumns
.macro file / .mf	.xhyphenate / .xhy
.over char / .oc	.xitalics / .xit
.paging / .pg	.xpaging / .xpg
.preface / .pf	.xpreface / .xpf
.prefix page / .pp	.xprefix page / .xpp
.print ptoc / .pptoc	.xsub
.print toc / .ptoc	.xsup
.right margin / .rm	.xul

A document is displayed on the screen or sent to the spooler by means of the OP command. This has the general form:

```
OP filename itemlist {(options}
```

where filename and itemlist identify the document(s) to be output and the options include any of:

m	output just page m
m-n	output pages m to n inclusive.
C	suppress CHAIN and READ commands.
D	double-spacing, and
T	triple-spacing.
H	suppress HEADING commands.
J	suppress HILITE commands.
S	suppress boldface, underlines, italics and fonts.

14 Access

There are a few extensions to the Access enquiry language under *Advanced Pick.*

* LIST ... BY is now identical to SORT ... BY

  Note that this does not yet apply to SELECT and SSELECT.

* There are new verbs for handling select-lists and saved-lists:

  + COMPARE-LIST
  + SORT-LIST

* The ISELECT and the ISSELECT verbs select all the items which are in the same physical group as a specific item and have the form:

```
ISELECT file.name item.id
ISSELECT file.name item.id
```

* The NSELECT verb produces a list of items in the current select-list which are not is a given file. A typical usage might be:

```
SELECT FILE1
NSELECT FILE2
```

  to produce a select-list of all those items which are on FILE1 but not on FILE2.

* There are several synonyms for the familiar verbs for handling select-lists and saved-lists:

  + CL for COPY-LIST;
  + DL for DELETE-LIST;

  + EDIT-LIST invokes the standard line-editor to edit a saved-list, whereas:

  + EL invokes the Update Processor to edit a saved-list.

  + GL for GET-LIST;
  + LL for LIST-LISTS;
  + SL for SAVE-LIST;

* There is a new list-processing verb: FL. This has the form:

```
FL file1 list1 {operators} {file2} list2}
```

  and allows lists held on two files to be combined by the set operators:

  = intersection, to build a list of items in both lists;

  + union, to build a list of items in either list;

  - exclusion, to build a list of items in list1 but

not in list2.

The result is produced as a select-list which can be used in the familiar manner.

The syntax of Access sentences is much the same as before, although there are some new keywords and conventions:

* Apostrophes may be omitted when specifying item-ids in an Access sentence. Thus, the forms:

```
LIST STOCK 1000 2000 DESCRIPTION
SORT STOCK > 2000 COLOUR PRICE
T-DUMP PROGRAMS LIST.WAGES PRINT.WAGES
```

are equivalent to:

```
LIST STOCK '1000' '2000' DESCRIPTION
SORT STOCK > '2000' COLOUR PRICE
T-DUMP PROGRAMS 'LIST.WAGES' 'PRINT.WAGES'
```

Care must be taken in the use of apostrophes if a file has any items and attribute-definitions with the same item-ids. If there is any conflict, a word is assumed to be an attribute-definition.

Note that it is still necessary to use quotation marks when specifying values in selection criteria:

```
LIST STOCK WITH COLOUR "RED"
```

* The maximum length of an Access sentence was 140 characters in length on R83. On Advanced Pick, there is virtually no limit on the length of the sentence which you may type in. The continuation sequence which allowed you to type in a long sentence in short segments has been discontinued.

* The FILL modifier (and the equivalent R option) may be used with an Access sentence which is too wide for the screen (or printer) and which would otherwise produce non-columnar output. When this is used, the data is output with the headings and the data continuing across the page in a text-like manner.

```
STOCK : 2000    DESCRIPTION   SETTEE, YELLOW, OAK
DATE   04 DEC 89    QTY   18

STOCK : 3000    DESCRIPTION   SIDEBOARD, BLUE, ASH
DATE   25 DEC 89    QTY   58

STOCK : 4200    DESCRIPTION   SETTEE, GOLD, ASH
DATE   14 DEC 89   27 DEC 89    QTY   55   10
```

* The DATA-ENTRY modifier allows a series of new items to be created by means of the Update Processor.

* The DUPLICATE modifier can be used with files and attributes for which an index is available. This is discussed in a

separate section.

* The file-name is included in the default heading for Access reports.

* There are additional options with HEADING and FOOTING modifiers.

* The ID-PROMPT modifier causes the Update Processor to ask for the item-id and allows the item-id to be edited, thereby leaving the current item and moving to another when using the Update Processor.

* The IFNO connective has been added. This has the same meaning as WITHOUT, as in:

```
LIST STOCK IFNO QUANTITY
LIST STOCK WITH NO QUANTITY
LIST STOCK WITHOUT QUANTITY
```

* The relational operator IS is offered as a synonym for =, EQ and EQUAL. Note that ARE is still a throwaway modifier.

* There is a new LEGEND facility which allows the user to specify a piece of text which is to appear at the foot of every Access report which is sent to the printer. The text is specified (in Update Processor format) in an item with the item-id LEGEND which is held on the MESSAGES, file.

  The inclusion of this message in Access reports is enabled/disabled by the LEGEND-ON / LEGEND-OFF commands and the LEG-SUPP modifier (or the equivalent K option) in Access sentences.

  A legend for a specific user may be placed on the MESSAGES file with the item-id:

```
LEGEND*user.id
```

* The XXX ITEMS LISTED message has been changed so that it shows how many items were listed and how many items are on the file.

* The NI-SUPP modifier (or the equivalent B option) suppresses the XXX ITEMS LISTED message at the end of an Access report. The equivalent S option (available on R83) has been withdrawn.

* The ROLL-ON modifier (and the synonym TOTAL-ON) rolls the output data into the following column, the previous column or as a new column. This is discussed in a separate section.

* The SAMPLING modifier will enable you to restrict your report to a specific number of items on the file. For example, the sentence:

```
LIST STOCK SAMPLING 20
```

  will display only the first twenty items on the file. If selection criteria are included in the sentence, as in

```
SORT STOCK WITH COLOUR "RED" SAMPLING 20
```

then these are evaluated before the sampling of the first twenty items takes place, reporting 20 red items.

* The SS - spreadsheet - connective produces a tabular summary of the values of one attribute between specific dates held in another attribute. This is discussed in a separate section.

* When an Access sentence produces a report *on the printer,* the sentence is normally shown at the head of the report. The action may be disabled/enabled for all Access reports by means of the TCL-HDR-ON and TCL-HDR-OFF commands and, for an individual report, by the TCL-SUPP modifier. The simple TCL-HDR will display the current status of the TCL headers.

* Attributes 11 and above of the system definition items (user-definition items, file-definition items, data-level identifier items, attribute-definition items) are used for a of purposes.

On R83 systems, an Access sentence such as:

```
LIST STOCK
```

with no output list, will automatically include numeric attribute-names 1, 2, 3 and so on, in the sentence, if these are available; this set of numeric attribute-names comprise what is known as the implicit output list. This is still supported on Advanced Pick but there is a further extension:

* Attribute 15 of the data-level identifier item may hold a list of attribute-definitions which are to be used in the absence of any other output specification. This *macro* specification is also used by the Update Processor. Any list held here is used in preference to the attribute-names 1, 2, 3, 4 and so on.

The macro may include the temporary attribute items described below.

The ONLY modifier suppresses the use of the macro and displays only the item-ids.

* A facility known as *temporary attribute items* which allow the use of attribute names such as A1, A2, A3 and so on in Access sentences. These are similar to the *A1, *A2 and *A3 definitions of R83 except that attribute-definition items A1, A2 and A3 do not exist, they are constructed for temporary use by the Access processor. The A1 definitions allocate the maximum possible width to the field in order to fit all the fields on the screen (or the printed page). For compatibility, the *A0, *A1 and other names are still available.

If the file dictionary contains explicit datanames A1, A2 and so on, these will be respected and will take precedence over the temporary attribute items.

14.1 ROLL-ON modifier

The ROLL-ON modifier *rolls* the output data into the following column, the previous column or as a new column.

The format of the ROLL-ON modifier is the same as the BREAK-ON modifier discussed in section 4.9:

```
ROLL-ON attribute "message"
```

where the *message* consists of any string of text and/or options - the options being enclosed in apostrophes. The options are the same as for the BREAK-ON modifier.

The annotated output below illustrates how several values may be rolled into one column, each being identified by a string of asterisks.

```
:SORT STOCK DESCRIPTION ROLL-ON PRICE ROLL-ON MINIMUM

STOCK..... DESCRIPTION.............

1500       CHAIR, BLUE, LAMINATE        <== DESCRIPTION
           * 60                         <== MINIMUM
           ** 25.00                     <== PRICE

2234       CHAIR, GREEN, OAK
           * 30
           ** 10.00

9555       TABLE, RED, PINE
           * 25
           ** 50.00
```

If the column to the left of the ROLLed-ON attribute has been modified by a TOTAL or other modifier, then the data will be rolled into the column to the right. If the column to the right has also been modified, then a new column will be created to hold the data.

14.2 SS - spreadsheet modifier

The SS - that is, SpreadSheet - connective produces a tabular summary of the values of one attribute between specific dates which are held in another attribute. For example, the sentence:

```
SORT STOCK SS DATED "15/11/92" "20/11/92" QTY
```

produces a table (spreadsheet) with columns headed 15/11/92, 16/11/92, 17/11/92 and so on up to "20/11/92" showing the total of the QTY attribute for those items which have a DATED attribute on the appropriate dates.

```
15/11/92 16/11/92 17/11/92 18/11/92 19/11/92 20/11/92 Total

      55       12       89      104       82       99   441
```

The output is euiqvalent to the control-break subtotals produced by a sentence such as:

```
SORT STOCK BY DATED BREAK-ON DATED TOTAL QTY WITH DATED
     >= "15/11/92" AND <= "20/11/92"
```

The general form of the SS clause is:

```
SS date.field "start.date" "end.date" total.field
```

The reference literature indicates that, if the end.date, 20/11/92 in our example, is omitted, then the current date is used, and that, if the start.date, 15/11/92 in our example, is also omitted omitted, then the processor will generate enough columns to fill the width of the report.

The G option may be specified to suppress row and column totals.

14.3 Attribute-definition items

On Advanced Pick, the attribute-definition item has the following structure:

```
001 dictionary-code
002 attribute-count
003 substitute-header
004 structure
005 not used
006 not used
007 output-conversion
008 correlative
009 attribute-type
010 column-width
011 not used
012 not used
013 not used
014 input-conversion
015 macro
016 output-macro
017 description
018 not used
019 not used
020 hotkey.all
021 hotkey1
022 hotkey2
023 hotkey3
024 hotkey4
025 hotkey5
026 hotkey6
027 hotkey7
028 hotkey8
029 hotkey9
030 hotkey0
```

Attributes 1 to 10 of attribute-definition items are used in

exactly the same way on previous versions, except that attribute 9 has additional codes:

9) Attribute 9: this specifies the justification of the output data. Extended codes are available on *Advanced Pick.*

Advanced Pick uses some of attributes 11 onwards. These extra fields are primarily concerned with the Update Processor:

14) Attribute 14: this specifies the input conversion.

15) Attribute 15: this specifies the macro - a list of attribute-definition names which are to be used when zooming to another item with the Update Processor.

17) Attribute 17: this is the description - a free text field for any comments concerning the use and nature of the attribute.

20) Attributes 20 onwards: these contain the CALLs to any subroutines which are to be invoked by the hot-key sequences.

These are discussed in detail in the following sections.

Attribute-definition items are created and modified by means of the UD - Update Dictionary - macro.

14.3.1 Attribute 9: TYPE or V/TYPE

Attribute 9 of D-pointer items and attribute-definition items is the TYPE or V/TYPE. This is a single letter specifying the justification of the output data, and indicates the way in which the output data is to be fitted within the column-width.

The extended codes on Advanced Pick are:

W indicates that, for non-columnar output, this attribute is to be processed as a piece of text by the Output Processor.

WW indicates that, for non-columnar output, this attribute and all subsequent attributes are to be processed as a piece of text by the Output Processor.

X is used on conjunction with L, R or T and indicates that the column for this attribute (and the associated column heading) is to be expanded to fill the width of the output report when this attribute is the right-most on the report.

If several fields on a report use the X code, then each will be expanded (in proportion to the column-wdith specified in attribute 10) so as to fill the width of the report.

For account-definition items (on the MDS file), this justification applies to the item-ids of the items on the MD of the account; for file-definition items (on the MD of the

account), this justification applies to the item-ids of the items on the DICT section; for data-level identifier items (on the DICT section of a file), this justification applies to the item-ids of the items on the data section;

4.3.2 Attribute 14: Input conversion

Attribute 14 of D-pointer items and attribute-definition items this specifies the *input conversion* - processing codes which are to be applied to the data immediately after it has been entered by the user under the control of the Update Processor. In general, the codes specified here can be any combination of the regular processing codes and the new input conversions implemented in *Advanced Pick*, and will be used to validate and transform the input data before it is written to the file.

If no input conversion is applied to a field, then - when using the Update Processor - the user must enter the data for that field in the appropriate internal form, although the data will be re-displayed in the correct external form.

4.3.3 Attribute 15: Macro

Attribute 15 of D-pointer items and attribute-definition items this specifies the *macro*. This is a list of attribute-definition items separated by spaces.

For data-level definition items, this is used as the *default output list* which is to be used if no output list is specified in an Access sentence. This macro list overrides the normal effect of attributes named 1, 2, 3 and so on.

The macro list in the data-level definition item is also used to indicate which fields are to be presented when items are modified by means of the Update Processor.

For attribute-definition items, this is a list of attribute-definition names which are to be used when this attribute is used to zoom to another item with the Update Processor.

4.3.4 Attribute 17: Description

Attribute 17 of D-pointer items and attribute-definition items this is the *description* - a free text field for any comments concerning the use and nature of the attribute. This text is used as a *help message* displayed if the user enters:

```
?
```

at any input field when using the Update Processor.

4.3.5 Attribute 20: Hot-keys

Attribute-definition items - and file-definition items - which are used by the Update Processor can exploit the *hot-key* feature whereby the user may invoke a Basic subroutine by hitting a hot-key.

A hot-key is a key sequence of the form:

<Ctrl> X n

where *n* is an integer 0 to 9, and the subroutines (if any) to be invoked by the hot-keys are declared in attributes 20 to 30 of the file-definition item and the attribute-definition item:

```
020 hotkey.all
021 hotkey1
022 hotkey2
023 hotkey3
024 hotkey4
025 hotkey5
026 hotkey6
027 hotkey7
028 hotkey8
029 hotkey9
030 hotkey0
```

If the user presses a hot-key sequence:

<Ctrl> X n

when entering a data field via the Update Processor, the processor will execute the first-encountered of:

1) the *hotkey.all* entry in the attribute-definition item for the data field;

2) the *hotkeyn* entry in the attribute-definition item for the data field;

3) the *hotkey.all* entry in the file-definition item;

4) the *hotkeyn* entry in the file-definition item.

A typical entry for a hot-key definition might be:

```
020 CALL SUB.HOTKEY.ALL.STOCK
```

or:

```
021 CALL STOCK.HOT.0
```

The Basic subroutine has the familiar form, with the following observations:

* No data is passed between the Basic subroutine and the data item being processed by the Update Processor.

* The ACCESS function is not available for capturing data from the item being updated.

* The subroutine must have:

```
SUBROUTINE subroutine.name(parameter)
```

as the first line although the *parameter* contains only a null value and is ignored outside the subroutine.

If several hot-key subroutines are to be called one after another, these may be specified in a form such as:

```
021 CALL STOCK.CHECK]CALL STOCK.HOT.0
```

separating the individual routine-names by value-mark.

15 Processing codes

* Advanced Pick introduces a number of additional processing codes:

```
B code - bridge correlative
CALL code - invoke a Basic subroutine
CU code - character update
I code - attribute index
I code - file index
I code - file reference
ID code - item-id
IF code - logical testing
MI code - mandatory input
MY code - hexadecimal to ASCII
O code - ordered multivalues
V code - value limitation
X code - prevent amendment
Xc code - update stamp
```

* Some of these are used in file-definition items and data-level identifier items, whilst others are used in as familiar Access correlatives in attribute-definitions items. In some cases, these new codes are provided for use in conjunction with the Update Processor and are held in the *input conversion* field (attribute 14) of the appropriate definition item.

* The D - date - code has a new form: DF to display the date in a form such as: July 29, 1993.

15.1 B code

The B - *bridge* - correlative is used in data-level identifier items and indicates that there is a relationship between an attribute on this file and an item on another file. It is a powerful means of ensuring data integrity within the database. Bridging is discussed in a separate section.

The B processing code will allow us to specify such a relationship between the files. We should use the correlative:

```
BCLIENT;1;3
```

in the data-level identifier item of the ORDER file to establish such a bridge to operate in both directions, from the ORDER file to the CLIENT and from the CLIENT file to the ORDER.

Whenever an item is added to or deleted from the ORDER file, attribute 3 of the CLIENT file is changed accordingly. In this situation, the Update Processor will not allow the order-numbers on the CLIENT file to be deleted until the corresponding order on the ORDER file has been deleted.

The full form of the B code is:

```
Bfilename;a;b{;c}
```

where *filename* indicates the file (the CLIENT file in our example) to which this file (the ORDER file) is associated (bridged); *a* is the attribute in the ORDER file which holds the item-ids of the CLIENT items; *b* is the attribute in the CLIENT file which forms the other end of the bridge.

The parameter *c* suppresses the bridge back to this (the ORDER) file implied by the parameter *b*, and has one of the forms:

n;+ meaning add the value of attribute *n* of the current ORDER item to attribute *b* in the CLIENT item,

n;- meaning subtract the value of attribute *n* of the current ORDER item from attribute *b* in the CLIENT item,

d meaning that the current ORDER item may be deleted even though there may be a value for attribute *a* of the item.

The following points are important when using bridges:

* When a new order is created on the ORDER file, the order-number will be added to the corresponding item on the CLIENT file.

* If there is no corresponding item on the CLIENT file, a new - empty - item will be created there.

* A bridge will prevent either the ORDER record or the CLIENT record being deleted by any Advanced Pick tool (such as the Update Processor or the EDITor) so long as there is a corresponding CLIENT or ORDER item on file.

* If, say, the client-number within the ORDER item is changed, the order-number will be removed from the original CLIENT item and added to the new CLIENT item.

15.2 CALL processing code

The CALL processing code is used to call a Basic subroutine from within an Access definition.

An example of the code might be:

```
CALL DRTN001
```

which will call a Basic subroutine catalogued with the name DRTN001.

Since this code, like all Access processing codes, can be used in the *input conversion* field of an attribute-definition item, Advanced Pick opens up the way to powerful data validation and transformation when used with the Update Processor.

When a definition using this code is encountered in an Access sentence, control is transferred to the subroutine:

```
SUBROUTINE DRTN001(INVALUE)
```

and a single parameter (INVALUE in this instance) is passed across containing the value derived (thus far) by the definition. Any processing may be done within the subroutine, including data input and output. When the subroutine processing is complete (and a Basic RETURN statement is encountered in the subroutine), the contents of the subroutine parameter (INVALUE) will be passed back to the Access processor and used for any further processing by other correlatives, the BY connective and the TOTAL modifier, before final output on the Access report.

Within the CALLed subroutine, other details of the file and the current item can be obtained by means of the ACCESS function.

15.3 CU code

Used as an input conversion in an attribute-definition item, the CU code allows character by character amendment of the data. The form is simply:

```
CU
```

Otherwise, when a user invokes the Update Processor to change this field on a file, the current contents of the field will be removed and replaced entirely by the new data which the user enters. However, if the definition includes this CU processing code, then the user may amend the existing data - character by character - and use the Advanced Pick cursor-control keys to skip over existing characters without changing them.

15.4 I code

Used as an input conversion in an attribute-definition item, the I code indicates that an index exists for this attribute. The form is simply:

```
I
```

When this attribute is being processed by the Update Processor, it allows the user to *cruise* through the entries in the index by means of the Advanced Pick control keys.

15.5 I code

This is held in the data-level identifier item and may be established by the CREATE-INDEX command. Indexing is discussed in a separate section.

Alternatively, the user may apply the code explicitly by setting attribute 8 of the data-level identifier item to:

```
IA2(G0 1)
```

indicating that index entries are to be maintained (in this case) for the second word of attribute 2. When a CREATE-INDEX command is next issued for this file, then the index will be established and this I code will be replaced by an entry of the form:

```
I98765A2(G0 1)
```

holding the frame address (98765 in this instance) of the root node of the appropriate index. Once the frame address of the root node has been set by the CREATE-INDEX command, it must not be changed by the user.

Whenever an item is added, changed or deleted from this file, the index will be maintained by the operating system.

Whenever any sorting or SSELECTion is performed on a field which is defined by a attribute-definition using this same A correlative, then the index will be used in performing the sort.

15.6 I code

Used as an input conversion in an attribute-definition item, this form of the I code indicates that the contents of this attribute are based upon the data which is indexed in another file.

For example, when we are using the Update Processor to enter data into the ORDER file, one piece of data which will be required is the client-number. When we are asked to enter the client-number, we should like to be allowed to enter the client *name* (or a part of it) and then get the system to look through the index of client names until we find the correct one and then use the client-number of that client as input data for the new ORDER item. This process of browsing through an index to another file is known as *zooming*.

In this situation, a typical instance of the I code used in the ORDER file might be:

```
ICLIENT;A1
```

which - when placed in the input conversion of the definition for the client-number on the ORDER file - will allow us to *zoom* across to the CLIENT file and browse through the index of attribute 1 (the name) of the CLIENT file whilst editing the client-number field of the ORDER file.

There should also be a Tfile processing code, such as:

```
TCLIENT;C;;1
```

in the attribute-definition for the client-number field of the ORDER file to permit the user to cruise through the CLIENT file.

When asked for a client-number, the user can zoom across to the index for the CLIENT file by means of the Advanced Pick controls keys.

15.7 ID code

Used as an input conversion in a data-level identifier item, the ID code specifies the manner in which the Update Processor is to create new item-ids as items are added to the file. There are several forms of the code.

ID with no parameters, the code will create a unique item-id of the form:

```
dn
```

where *d* is the internal date and *n* is a unique sequence number.

IDAexpression
: will use the A (correlative) expression to derive the item-id from the other data in the item.

IDn
: will create unique numeric item-ids starting at *n*. As new items are added to the file, the system will update the value of *n* in this code to reflect the next number to be used.

As new item-ids are generated under the control of this code, if it is found that an item already exists with the generated item-id, then the system will increment the generated item-id until a unique item-id is obtained.

Furthermore, if the code is currently set at, say:

```
ID500
```

Then any gaps in the sequence of item-ids lower than 500 will be ignored. Such gaps can be filled in by resetting the code to:

```
ID0
```

IDT will create an item-id of the form:

```
dt
```

where *d* is the internal date and *t* the internal time.

15.8 IF code

The IF processing code is used to test one or more conditions and output an expression if the result is true, or another expression in the result is false.

The general format of the code is:

```
IF condition THEN expression
IF condition ELSE expression
IF condition THEN expression ELSE expression
```

where *condition* is any conditional expression and *expression* is any output expression (which may include a further IF code).

Some examples are:

```
IF 7 < 2 THEN "BELOW" ELSE "NOT BELOW"
```
will test the contents of attribute 7 of the data item, and output the word BELOW if the value is less than the contents of attribute 2, otherwise the word NOT BELOW will be output.

```
IF 2+"0" = "0" THEN 5 ELSE 2
```
will test the contents of attribute 2 of the data item; if the value is zero or null, then the contents of attribute 5 will be output, otherwise the contents of attribute 2 will be output.

```
IF 2 = 5 THEN "OK" ELSE IF 2 < 5 THEN "LOW" ELSE "HIGH"
```
will output one of the words OK, LOW or HIGH according to the relative contents of attributes 2 and 5.

```
IF 1 = "0" THEN "ZERO VALUE FOUND"
IF 1 = "0" ELSE "NON ZERO VALUE"
IF 1 > 2 THEN 1 ELSE 2
IF 2+4 > 7 THEN 2+4 ELSE 7
IF 1 AND 2 THEN 1+2 ELSE "0"
IF 1 OR 7 THEN "OK" ELSE "ERROR"
IF 1 AND 7 THEN IF 1 > 7 THEN 1 ELSE 7 ELSE "0"
IF N(VALUE) > "1000" THEN "REVIEW" ELSE N(VALUE)/100
IF 1 THEN IF 2 THEN 3 ELSE 4 ELSE 5
IF N(S0) < "100" THEN N(S1) ELSE N(S2)
IF (IF 1 THEN 2 ELSE 3) THEN 4

IF N(CODE)R = "2" THEN (N(QTY)*N(PRICE2)R)(MD2) ELSE
     IF N(CODE)R = "3" THEN (N(QTY)*N(PRICE3)R)(MD2)
     ELSE (N(QTY)*N(PRICE1)R)(MD2)
```

Normally, the code will comprise matching sets of IF/THEN/ELSE clauses, but the END keyword may be used for situations where an ELSE is not to be paired with the latest IF:

```
IF 1=2 THEN IF 3=4 THEN 5 END ELSE 6
```

in which the ELSE 6 pairs with the first IF and the first

THEN, not with the THEN 5. Without the END, the code would be interpreted as:

```
IF 1=2 THEN IF 3=4 THEN 5 ELSE 6
            └──────┴──────┘
└──────┴──────────────────────┘
```

The following report shows the output from these codes (used in definitions called CODE1 and CODE2, respectively), for various values of attributes 1, 2, 3 and 4.

```
>LIST TESTER *A1 *A2 *A3 *A4 CODE1 CODE2

TESTER.... Attr1 Attr2 Attr3 Attr4 CODE1 CODE2

1            111   222   333   444   666
2            111   111   333   444         666
3            222   222   333   333   555   555
4            111   222   333   333   666
```

15.9 MI code

Used as an input conversion in an attribute-definition item, the MI code specifies mandatory input, that is, that the user must enter a data value into the field when using the Update Processor. The form is simply:

```
MI
```

If the user attempts to enter null for such a field, then the system will beep and return to input the data.

15.10 MY code

This converts hexadecimal data into ASCII character format. The form is simply:

```
MY
```

15.11 O code

Used as an input conversion in an attribute-definition item, the O code specifies that, as they are added to the file by the Update Processor, the multivalues of this field are to be sorted into left-justified ascending order. The form is simply:

```
O
```

15.12 V code

Used as an input conversion in an attribute-definition item, the V code specifies a maximum number of multivalues which are allowed in this field when using the Update Processor. Sorted into left-justified ascending order. An instance might be:

```
V5
```

which will only permit 5 multivalues to be entered. The special form:

V0

Prevents the user from entering *any* data into the attribute.

15.13 X code

Used as an input conversion in an attribute-definition item, the X code specifies that the field cannot be amended by means of the Update Processor. The form is simply:

X

15.14 Xc code

Used as an input conversion in a data-level identifier item, this form of the X code specifies a *stamp* that is to be set in a specific attribute of each item as it is updated. There are several forms of the code:

XAn
: will put the name of the user in attribute number *n* of the updated item.

XDn
: will put the current date in attribute number *n*.

XSn
: will accumulate in attribute number *n* the number of seconds for which the item was handled by the Update Processor.

XTn
: will put the current time in attribute number *n*.

If a further element *V* is specified - XAnV, XDnV, XSnV, XTnV - then the stamp will be *appended to the stamp attribute as a new multivalue*, otherwise the stamp will replace the previous contents of the attribute.

16 File indexing

Any item on a Pick file can only be retrieved if you know the name of the file and the item-id of the specific item. For example, the items on the STAFF file will be uniquely identified by the item-id, the employee number. However, there may be an operational requirement to find an item if you know the employee's surname or the department in which the employee works. Such secondary information, the surname or the department, will probably not be unique and is known as a *secondary key* into the file. Advanced Pick allows you to create an *index* which will offer faster access to the data on your files by way of such a secondary key. A file may have any number of such indexes, one for the surname, one for the department, one for the telephone number, and so on. In fact, the index (some people use the term *B-tree*) may be based upon any information derived from the item. The *instructions* for deriving the index-key are specified as A processing codes, and the operating system stores these instructions together with the FID of the root of the index in attribute 8 the data-level definition item for the data section of the file. They will be held in the form of an I-code, such as:

```
I03DBF4A3(G1 1)
```

This particular instance shows that the root FID of the index which is based on the A processing code:

```
A3(G1 1)
```

is to be found in frame 03DBF4 (= frame 252916 in decimal). The index is held in frames of virtual memory, not on a Pick file.

If the file has more than one index, each index will have its own I-code in the data-level definition item for the file. These are organised as multivalues of attribute 8.

The index is maintained automatically by the operating system whenever items are added, modified or deleted from the main data file by any of the standard Advanced Pick processors, including:

* The Update Processor,
* The COPY verb,
* The T-LOAD verb.

Apart from its value in file-access, a major application of the index is in *cruising*, that is scanning backwards/forwards through the file using the specified field as a key. Thus, if we have an index for the employee's surname, we can cruise through all the SMITHs until we find the one we want.

The CREATE-INDEX command is used to create a new index for a file. The general form of the statement is:

```
CREATE-INDEX filename code
```

where *filename* is the name of the file for which the index

is to be created, and *Acode* is the A processing code which is to be used. Here are some examples:

CREATE-INDEX LIBRARY A0
will create an index to the file LIBRARY, the keys for the entries in the index consisting of the item-id of the items on the LIBRARY file.

CREATE-INDEX STAFF A3(G1 1)
will create an index for the STAFF file, the keys consisting of the second word of attribute 3 (this might well be the surname) of the items on the STAFF file.

In each case, the CREATE-INDEX verb will establish a B-tree index with the nodes held in frames of disk space (not as items on a file), and it will then go on to set up an I-code for the index in the data-level identifier item for the relevant file, as described above.

A number of Basic statements and other facilities are provided to support indexing.

* The VERIFY-INDEX command checks the integrity of an index.

* The NFRAME-INDEX command displays the number of frames consumed by index.

* The DELETE-INDEX command deletes a specific index, removing the index and the I-code.

If there is an index for the sort keys of a SORT or SSELECT sentence, then these will be used to speed up the action of the sort. Thus, if we issue an Access sentence such as:

```
SORT STAFF BY SURNAME
```

and the processing code in the attribute-definition item for SURNAME matches any of the existing A processing codes for an index for the STAFF file, then this index will be used to retrieve the sorted data.

The DUPLICATE modifier may be used in conjunction with the WITH modifier in a context such as:

```
LIST STOCK WITH DUPLICATE LOCATION DESCRIPTION LOCATION
```

to list only those items which have duplicate index keys for the LOCATION field, that is, where two or more items have the same LOCATION field.

The Basic ROOT statement is used to find the FID of the root of an index for a file and allow the program to access items by means of the index. The general form of the statement is:

```
ROOT filename, acode TO root THEN / ELSE
```

where: *filename* is the name of the file for which the index is to be used; *acode* is the A processing code which is used to maintain the index; *root* is the variable which is to hold

the FID of the root, this will be used in subsequent KEY statements to retrieve item-ids from the index, rather like a normal file-variable.

The Basic KEY statement allows the program to access items by means of the index established for the file. The general form of the statement is:

```
KEY(code, root, key, id) THEN / ELSE
```

where: *code* is any of

C to return the closest match to the *key*,

N to return the next key to *key*,

P to return the previous key,

R to read the key and return the corresponding item-id.

root is a variable which contains the FID of the root of the index, and is set by means of the ROOT statement; *key* is a variable which contains the index-key which the program is passing to the routine which scans the index; *id* contains the item-id of the required item. The ELSE clause is executed if the required key cannot be found.

17 Basic

The Advanced Pick implementation differs from previous versions of Basic in the following important areas:

* It is not necessary to have the program file defined with a DC-pointer. A Basic program can be held and compiled on any type of file.

* Any Basic programs transferred from R83 to AP must be recompiled before they can be used on AP.

* The format of catalogued-pointers on the MD has changed.

* Blank lines may appear within the source program. Previously, there had to be at least an asterisk there to placate the compiler. Labels without a following statement are also permitted:

```
DESTINATION:
10
```

* There are two verbs to compile a Basic program:

  + BASIC which is identical to previous versions in which variable names are *case sensitive* (that is, FRED and Fred are different variables), and

  + COMPILE in which variables names are not case sensitive.

* On release 6.0 and higher which is offered for use on Unix systems, there is a new version of Basic, known as *flash Basic.* This compiles directly into object code and is therefore significantly faster than standard Pick Basic code which is interpreted at run-time.

* A number of additional verbs are available: COMPILE to compile a Basic program in which the variables and the Basic keywords are entered in upper- and/or lower-case; COMPILE-RUN to compile and then execute a Basic program; COMPILE-CATALOG to compile and then catalog a Basic program or subroutine.

* The B/LIST command is used to indent/format a Basic source program.

* The RUN-LIST command will execute, one after another, a sequence of programs submitted as a select-list. This might be used in a sequence such as:

```
SSELECT WAGES.PROGRAMS = "YEAR.END]"
RUN-LIST WAGES.PROGRAMS
```

Many of the features of Basic on Advanced Pick - such as the IN and OUT statements - will be familiar to programmers who have worked with Release 3.1 PC Pick.

* The ^ operator and the ** operator are used for exponentiation.

* The \ operator can be used to return the remainder after

integer division. Thus:

```
A\B
```

is equivalent to MOD(A,B).

* The literature tells us that a feature, known elsewhere as *operator assignment*, is available. This accepts the following notation:

Statement	Equivalent
AAA += BBB	AAA = AAA + BBB
AAA -= BBB	AAA = AAA - BBB
AAA := BBB	AAA = AAA : BBB
AAA *= BBB	AAA = AAA * BBB
AAA /= BBB	AAA = AAA / BBB
AAA \= BBB	AAA = AAA \ BBB

On my 5.2 version, these forms would not compile.

* The range of @ function calls has been extended.

* On those systems which work alongside DOS or Unix environments, there is a range of statements - of the form %xxxx - which will communicate with those environments and handle their files.

* Substring assignments can be used to insert a substring within a string. Thus, the sequence:

```
STRING='ABCDEFGHIJKLMNOPQ'
STRING[2,3]='PQRST'
CRT STRING
```

would display: APQRSTEFGHIJKLMNOPQ

More complex is the group-store version. This resembles some aspects of the Access Group-extraction code and replaces one or more groups of data within a string. Thus, the sequence:

```
STRING = "A,B,C,D,E,F,G"
STRING[",",2,3] = "X,Y,Z"
CRT STRING
```

would display: A,X,Y,Z,E,F,G

* The ACCESS function has been provided to access data within subroutines which are called by Access attribute-definitions.

* The ASSIGNED function can be used to determine whether or not a variable has been assigned a value. This is useful in subroutines which are called by user programs and where data parameters may or may not have been set up correctly before calling the subroutine.

* The CASING ON/OFF statements can be used to specify whether keyboard input data is to be case sensitive or not.

* The CLOSE statement has been provided to close a file which is no longer to be accessible to the program.

* Labelled COMMON blocks of data, such as:

```
COMMON /STAFF/ NAME, AGE, RATE, CLOCK.NO
```

  are available for use in subroutines which only handle a subset of the COMMON data.

* The CONVERT statement translates the contents of a string acccording to two control strings.

* The DEL and INS statements are offered for handling dynamic arrays, as other many non-Pick systems. The statement:

```
DEL REC<1,2,3>
```

  is equivalent to the standard:

```
REC=DELETE(REC,1,2,3)
```

  and the statement:

```
INS 'ABC' BEFORE REC<1,2,3>
```

  is equivalent to the standard:

```
REC=INSERT(REC,1,2,3; 'ABC')
```

* When handling dimensioned arrays, logical expressions may use the form:

```
arrayname(*)
```

  in a statement such as:

```
IF LIST(*) = '0' THEN ... ELSE ...
```

  to take a specified action if *any element of the dimensioned array* satisfies a certain condition.

* The ERROR statement can be used to display a message from the MESSAGES (or ERRMSG) file. Unlike the STOP statement, which also outputs a message, the ERROR statement does not terminate the program.

* The ERROR() function returns the TCL command which invoked the program. Thus, the statement:

```
PRINT ERROR()
```

  is similar to the sequence:

```
TCLREAD DUMMY ELSE STOP
PRINT DUMMY
```

* The EXIT statement leaves the current loop by skipping to the statement immediately following the next physical NEXT or

REPEAT statement.

* Under certain implementations, the EXECUTE statement can be used to invoke commands in the host environment, such as DOS and Unix commands.

* The FILE statement is used to open a file, declare a suitable dimensioned array and accept data references of the form STOCK(QUANTITY) to use Access dictionary definitions in the coding of a Basic program.

* The FOLD statement breaks a string down into substrings of a specific length.

* The GET statement gets a string of ASCII characters from a specific port. The port must have been attached by means of the DEV-ATT command. The associated GETX statement returns a hexadecimal string.

  The SEND and SENDX statements will send a string to a port.

* The HEADING/FOOTING statement has an extended range of options.

* IN and OUT are provided for single character I/O, as on PC Pick.

* The IN and INPUT statements have extended forms:

  IN variable {FOR time {THEN / ELSE statements}}

  INPUT variable { ,length} {:} {_} {FOR time {THEN / ELSE statements}}

  In all cases, the *time* is the limit (expressed in tenths of a second, and in the range 1 to 32767) for which the processor will wait for the user's input. If data is entered within this time limit, THEN clause is taken; if no data is entered with the time limit, the ELSE clause is taken.

* $INCLUDE is offered as an alternative to INCLUDE.

* LET is optional on assignment statements.

* MATBUILD statement uses the contents of a dimensioned array to construct a dynamic array:

  MATBUILD dynarr FROM dimarr USING char(254)

* MATPARSE statement uses the contents of a dynamic array to load a dimensioned array:

  MATPARSE dimarr FROM dynarr {USING char(254)} SETTING count.variable

* The simple assignment statement can also be used to transfer data - element by element - between a dimensioned array and a dynamic array. Thus, the statement:

  SALES.DIM = SALES.STORE

will transfer the individual elements of the dynamic array SALES.STORE to the dimensioned array SALES.DIM (which is declared in the normal manner), and the statement:

```
SALES.STORE = SALES.DIM
```

will use the elements of the dimensioned array SALES.DIM to construct a dynamic array in the variable SALES.STORE.

* The OCCURS function searches a specific string for substrings which occur a specific number of times.

* The statements which handle the backing storage device may specify the ONERR clause. This is similar to the ELSE condition except that ONERR places a code in SYSTEM(0) indicating the nature of the error. ONERR and ELSE cannot be specified in the same statement.

* The THEN/ELSE clauses may be omitted on the OPEN statement. A form such as:

```
OPEN 'STOCK'
```

is equivalent to the standard form:

```
OPEN 'STOCK' ELSE STOP 201,'STOCK'
```

* The PRECISION statement will allow a precision in the range 0 to 9. The default is still 4.

* The THEN/ELSE clauses may be omitted on the various READ statements. A form such as:

```
READ S.REC FROM ITEM.ID
```

is equivalent to the standard form:

```
READ S.REC FROM ITEM.ID ELSE S.REC=''
```

* The READTL statement will read the label on the backing storage device. This has the form:

```
READTL variable THEN / {ELSE / ONERR statement(s)}
```

* The READTX command will read a record from backing storage and pass this to the program as a hexadecimal string. This means that records which include Pick and Basic control characters - such as CHAR(255) - can be processed.

The equivalent WRITETX is not available.

* The REPLACE statement is used within a bridge link to change an item-id.

* The ROOT statement finds the root of a B-tree index, and the KEY statement uses this to find within the index a specific item-id which matches a given argument. The KEY statement has options to return the item-id which matches the argument, the next sequential item-id, or the previous item-id,

* The SORT function sorts the elements of a dynamic array.

* The SOUNDEX function returns a four-digit code equivalent to a specified string. This has the form:

```
CODE=SOUNDEX(NAME)
```

or:

```
CODE=SOUNDEX(NAME,TYPE)
```

where TYPE is 0 (for R83 compatible census codes), or 1 (for standard English soundex codes).

* The SUM function will add together will add together all the adjacent subvalues of a dynamic-array and replace these with the result as a single value; if there are no subvalues, then it will add together all the adjacent values to produce a single attribute; if there are no adjacent values, then it will add together all the attributes to return a single numeric result.

* The range of SYSTEM function calls has been extended.

* The TCL statement execute a specific TCL command and, optionally, returns the error-message numbers generated by the command to a dimensioned array in the program. A sequence of several TCL commands may be invoked if the individual commands are separated by attribute-marks. The format of the statement is:

```
TCL tcl.command {to dimensioned.array}
```

The general effect of the TCL statement is similar to that of the EXECUTE statement.

* The TCLREAD statement will pass the TCL command which invoked the program into a specified variable.

The C language receives special attention within AP/DOS and the various AP/Unix versions where C functions may be invoked within Basic programs.

17.1 ACCESS function

The CALL processing code in an Access attribute definition passes the current data value to a CALLed subroutine as a single argument. Other details of the file and the current item can be obtained within the subroutine by means of the ACCESS function.

The ACCESS function takes a single numeric argument, each of which returns a specific piece of information.

The arguments for the ACCESS function include:

1 returns the file-variable for the data section of the file. This may be used in READ statements within the subroutine without having to execute an OPEN statement.

2 returns the file-variable for the DICT section of the

file. This may be used in READ statements within the subroutine. The DICT section is a useful place to store accumulators and control totals generated by the subroutine.

3 returns the entire contents of the current item as a dynamic-array. This is useful in extracting further data from the item. This will be null if this is a new item.

4 returns the current item counter, that is, the number of items processed thus far by the Access report. In addition to its use in calculating averages and the like, this can also be used to clear and maintain running totals (for example, it might be appropriate to clear an accumulator when this has a value of 1).

5 returns the current attribute counter, that is, the number of the attribute which is being processed by the attributed definition which called the Basic subroutine.

6 returns the current value counter.

7 returns the current subvalue counter.

8 returns the current detail-line counter, that is, the number of lines printed since the last control-break.

9 returns the current control-break level counter.

10 returns the current item-id.

11 returns the name of the file.

12 returns 1 if the item is being deleted.

13 returns the root variable when used with files for which an index is being maintained by the operating system.

14 returns the current cursor column position, when used with the AP Update Processor.

15 returns the current row column position, when used with the AP Update Processor.

16 returns 1 if the item is new.

17 returns 1 if performing an input-conversion.

18 specifies which value of the macro (attribute 15) is to be used.

19 returns the character which terminated the last keyboard input.

20 returns 1 if the item has changed.

21 specifies a character string to be passed to the Update Processor.

References to the ACCESS function may be unacceptable in certain contexts. For example:

```
READ RECORD FROM ACCESS(1), 'CONTROL' THEN ...
```

must be paraphrased in a manner such as:

```
FILE.VAR=ACCESS(1)
READ RECORD FROM FILE.VAR, 'CONTROL' THEN ...
```

17.2 FILE statement

The FILE statement allows a Basic program to make greater use of the file dictionaries, as illustrated by this fragment:

```
* Process items on STOCK file

* Open the STOCK file

    FILE STOCK

* Messages and displays

* Main loop

    LOOP

        INPUT KEY

    UNTIL KEY = ESC DO
        MATREAD STOCK FROM FV.STOCK, KEY THEN
            PRINT HDR
            PRINT KMESS: KEY
            PRINT DMESS: STOCK(DESC)
            PRINT QMESS: STOCK(QTY)
            PRINT LMESS: STOCK(LOCN)

            INPUT CHANGE

            STOCK(QTY) = STOCK(QTY) + CHANGE
            MATWRITE STOCK ON FV.STOCK, KEY

        END
    REPEAT
```

The statement:

```
FILE STOCK
```

stimulates the compiler to perform a number of tasks:

+ Open the file called STOCK to the file-variable FV.STOCK.

+ Recognise identifiers of the form:

```
STOCK(DESC)
STOCK(QTY)
STOCK(LOCN)
```

as references to the Access *attribute-definition* items DESC, QTY and LOCN on the DICT section of the STOCK file.

\+ Set up a dimensioned array called STOCK. The references to the attribute-definitions for the STOCK file are checked by the compiler and the highest numbered attribute determines the size of the dimensioned array set up by the program. The dimensioned array is one more than the highest attribute referenced in order to handle any data attributes beyond those used in this program.

Since the details of the attribute-definitions are *bound* into the program when it is compiled, any changes which are made to the file dictionary and which affect these definitions will not be reflected in the program until it is recompiled.

17.3 HEADING / FOOTING options

In all contexts - Basic, Runoff, Access - the general form of the HEADING and FOOTING statement is unchanged, but there are additional options:

J right-justifies the following text.

N suppresses end-of-page pause on terminal output.

R outputs the current page-number in Roman numerals.

S outputs subsequent text in italics, if the output device supports this facility. XS will switch this off.

U underlines subsequent text, if the output device supports this facility. XU will switch this off.

V outputs subsequent text in boldface, if the output device supports this facility. XV will switch this off.

X has the combined effect of XS, XU and XV in switching off all special output effects.

XS switches off italic output which was invoked by the S options.

XU switches off underlined output which was invoked by the U options.

XV switches off boldface output which was invoked by the V options.

17.4 Dimensioned arrays

Some of the statements for handling dimensioned arrays have changed slightly. After the following statements:

```
     DIM DIMARR(5)
     STRING='A'
     DYNARR='A':CHAR(254):'B'
```

DIMARR=0
     will leave DIMARR containing

element 1	0
element 2	null
element 3	null
element 4	null
element 5	null

MAT DIMARR=0
     will leave DIMARR containing

     (the standard action)

element 1	0
element 2	0
element 3	0
element 4	0
element 5	0

DIMARR=STRING
     will leave DIMARR containing

element 1	A
element 2	null
element 3	null
element 4	null
element 5	null

DIMARR=DYNARR
     will leave DIMARR containing

element 1	A
element 2	B
element 3	null
element 4	null
element 5	null

and this followed by:

STRING=DIMARR
     will leave STRING containing:

```
          A:CHAR(254):B
```

Thus, the statement:

```
     DIMARR = DYNARR
```

is equivalent to the MATPARSE statement found on some systems, and the statement:

```
     DYNARR = DIMARR
```

is equivalent to the MATBUILD (or MATUNPARSE) statement found on some systems.

When reading data from a file, the forms:

```
     READ DIMARR FROM ...
```
and:
```
     MATREAD DIMARR FROM ...
```

are identical, provided that DIMARR has been declared as a dimensioned array. The same comments apply to the

WRITE/MATWRITE statements.

18 The spooler

The Advanced Pick spooler is essentially the same as on previous versions. The TCL commands which are associated with the spooler:

LISTPEQS	SP-EDIT	STARTPTR
SP-ASSIGN	SP-KILL	STOPPTR
LISTPTR	SP-OPEN	
SP-CLOSE	SP-TAPEOUT	

are unchanged and the LISTPEQS and other reports have their familiar forms:

```
Spooler Queue Elements.                              29 Jul 1993 08:30:59
Job Stat Lnk  Line Status   Copies Form Frames   Date     Time    User

  1 80C8         0   c h  p      1    0      1 29/07/93 08:56:59 dm
  2 8548   3     0   c    p      1    0     12 29/07/93 09:40:16 dm
  3 8548         0   c    p      1    0      1 29/07/93 09:40:57 dm

  3 Queue Elements.                          14 Frames in use.
```

These are described in detail in the *MB-Guide to the spooler*, and in the Advanced Pick Reference Manual.

A new command is:

```
ASSIGNFQ q p
```

which associates form-queue number *q* with printer *p*. There will be a definition for printer *p* on the DEVICES file.

19 File-saves

In general, the processes:

* file-save
* file-restore
* account-save
* account-restore
* selective restore (SEL-RESTORE)

are performed exactly as on R83 systems.

The following changes are of interest:

* The R83 and Advanced Pick T-DUMP / T-LOAD files are compatible in both directions, and R83 file-save / account-save files can be used for account-restore and/or sel-restore purposes on Advanced Pick, though Advanced Pick file-save / account-save files cannot be used to pass data to R83 implementations.

* The file-save will process a select-list of accounts to be saved. Thus, sequences such as:

```
SELECT MDS 'WAGES' 'PAYROLL' 'PAY.STATS'
FILE-SAVE
```

```
SELECT MDS 'WAGES' 'PAYROLL' 'PAY.STATS'
SAVE (DFT
```

will save just the accounts WAGES, PAYROLL, PAY.RATES together with the DM account.

* Irrespective of whether or not you use transaction logging, Advanced Pick offers an incremental file-save facility which, during a file-save operation, will dump only those groups which have been changed since the last complete system save (file-save).

The items which have been changed by any process have a flag set to indicate this. When an incremental file-save is performed, only these flagged items will be saved. The various options of the SAVE command may then un-flag these items.

* The FILE-SAVE command now asks:

Is this an incremental save?

and then proceeds in the familiar manner. The ACCOUNT-SAVE command is unchanged.

* There is a facility to save only those items whose item-ids are held in attribute 25 onwards of the item on the FILE-OF-FILES file which relates to the file being processed.

* The SAVE verb has new options associated with the incremental save facility:

U to save all items which have been changed since the last file-save, and then clear the changed-flags so that these items will not appear on subsequent incremental saves.

V to perform a full file-save/account-save but do not remove the changed-flag from those items which have been changed since the last file-save, so that the changed items will appear on subsequent incremental saves.

* The :FILES command is no longer used to perform a file-restore.

  The file-restore action can now be performed by taking the F option when the system is rebooted with a file-save / system disk loaded.

* The account-name which is specified on the ACCOUNT-RESTORE command will be assumed if a null response is given to the NAME OF ACCOUNT ON DISK message.

* The RESTORE-ACCOUNTS utility will read through a file-save diskette/tape and restore any accounts which are not on the system.

* The UPDATE-ACCOUNTS utility is available when transferring any master dictionaries from R83 systems to Advanced Pick, and this will update the verbs on the master dictionaries to the Advanced Pick level. The command has the form:

```
UPDATE-ACCOUNTS account.name {account.name}
```

20 Coldstart features

A number of coldstart macros are invoked whenever the system is started:

* SYSTEM-COLDSTART is held on:

  DM,MD, SYSTEM-COLDSTART

  and contains a number of general initiation commands.

  It will be worth inspecting this item and removing any, such as:

  CLEANPIBS
  SETPIB0
  VERIFY.SYSTEM

  which may reset any parameters which the user wished to set permanently and/or hinder your system.

* USER-COLDSTART is held on:

  DM,MD, USER-COLDSTART

  and should be amended to include any special routines which are to be invoked when the system is started.

Recall that:

* When an individual user logs on, any commands specified in the macro field of the USERS entry will be invoked, and

* When a user logs to another MD, any Logon Proc held on the MD with the name of the account (MD) will be invoked.

21 Summary charts

In this final section, we summarise the most important key sequences which are used with the Update Processor.

In this table, I have used the same notation as in the Advanced Pick Reference Manual for the control keys:

- Y representing the sequence <Ctrl> Y and:
- XF representing the sequence <Ctrl> X F and:
- M representing the sequence <Ctrl> M that is, the <Return> key.

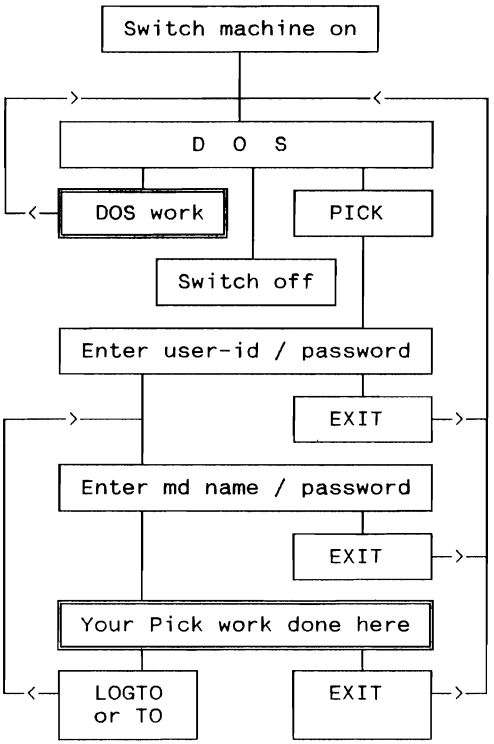
Key	TCL stacker
D	Go to next command in TCL stack
F	Go to previous command in TCL stack
X	Abandon the displayed command
	General cursor movement
B	Move cursor up one line
D	Move cursor left one sentence
F	Move cursor right one sentence
G	Move cursor to end of paragraph/attribute
I	Move cursor right to next tab stop
J	Move cursor left one character
K	Move cursor right one character
M	Move cursor to next paragraph/attribute
N	Move cursor down one line
P	Move cursor down one screen
T	Move cursor to top of item
U	Move cursor right one word
Y	Move cursor left one word
Z9	Move cursor to line 9
ZA	Redisplay screen with current line at top
ZB	Redisplay screen with current line at bottom
ZE	Move cursor to end of item
ZG	Move cursor to end of paragraph/attribute
ZH	Display with bottom line at line 11 of screen and cursor at line 12
ZN	Move cursor down one screen
ZP	Redisplay screen
ZQ	Display with bottom line at line 17 of screen and cursor at line 18
ZY	Move cursor up one screen
	General text editing
E	Delete to end of sentence
H	Overwrite character to left with a space
L	Delete one character
O	Delete to end of a word
R	Toggle overtype/insert mode
V	Insert a value-mark
W	Insert one space
ZC	Display column position mask

ZD	Delete to end of item
ZO	Delete to end of paragraph/attribute
ZT	Set tab stops
ZZ	Cancel last change
	General filing
XB	Return to the previous item
XC	File, compile and catalog the (program) item
XE	Abandon the item without filing
XF	File the item
XI	Rename and file the item
XK	Abandon the item(s) and return to TCL
XN	Exit item and go to a new item
XO	Exit and delete the item
XP	File and print the item
XR	File, compile and run the (program) item
XS	Save the item and continue
XX	Abandon the item without filing
	General Update Processor functions
D	Cruise to previous indexed item and edit
E	Get next sequential index
F	Cruise to next indexed item and edit item
G	Zoom to another file
U	Get next indexed item
Y	Get previous indexed item
	Cut and paste
CC	Mark end of text block
CD	Mark start of text block for deletion
CL	Mark start of text block for copying
CP	Paste in marked block
CRfiM	Read text from a file
CWfiM	Write current paste block to a file
	Pre-store command buffer functions
P	Execute command(s) in prestore buffer
ZL	Amend command(s) in prestore buffer
ZRfiM	Read prestore buffer from a file
ZRfiP	Read prestore buffer from a file and execute
ZWfiM	Write prestore buffer to a file
A	Repeat last search
	Search and replace functions
AxM	Search for string x
AxRyN	Replace all occurrences of string x by string y
AxRyM	Replace occurrences of string x by string y confirming changes
A	Ignore this occurrence and repeat the search
N	Replace all occurrences
R	Replace this occurrence and repeat the search
X	Abandon search

	Spelling checker functions
ZS	Disable the spelling checker
A	Ignore the word
AA	Accept the word and add to the WORDS file
AAA	Disable the spelling checker

22 Routes through the system

This diagram illustrates some of the possible routes through the system.



You can use DOS facilities from within AP/DOS, but we have not shown this on the diagram.

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
* Error Messages
* File design
* File-save & file-restore
* Files: file sizing tables
* Files: monitoring & sizing
* Group format errors
* Jet word processing
* Operations & systems management
* Pick fundamentals
* Pick on the PC
* Pick terminology
* Producing training courses
* Program design
* 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

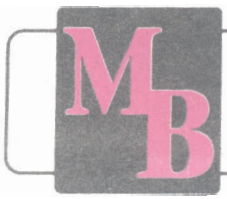
MB-Guides

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 operating system.



Malcolm Bull

Training and Consultancy Publications