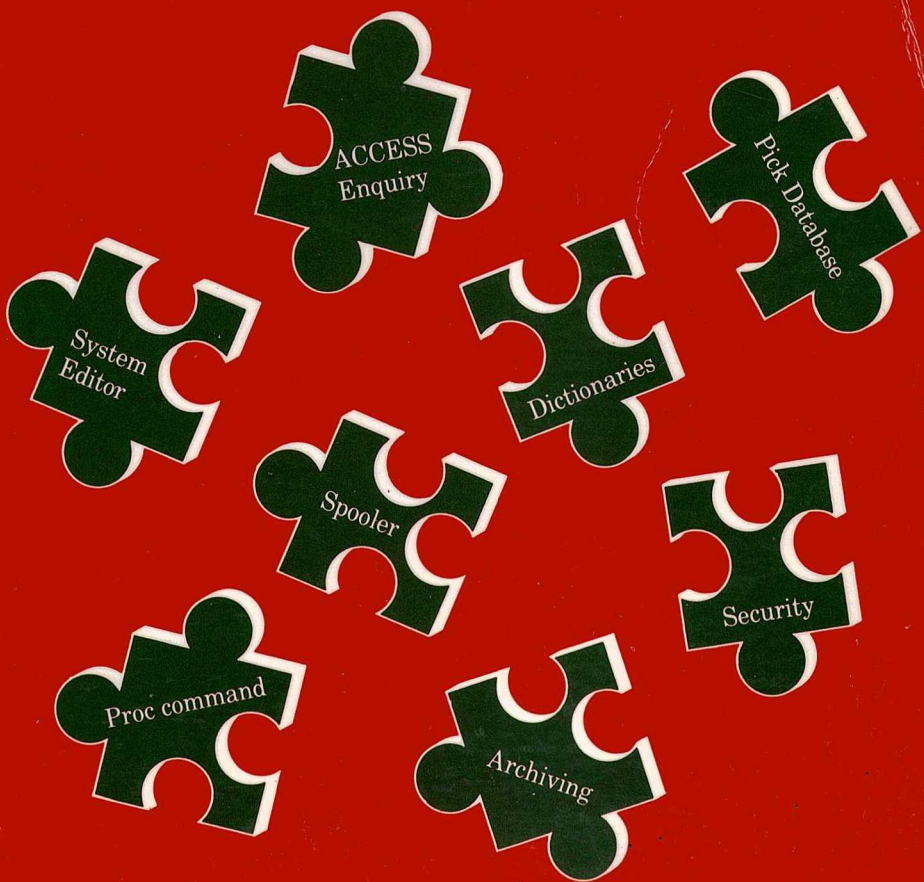
# PICK™

## for users

Martin Taylor




Blackwell Scientific Publications

# PICK™ for users

# PICK™ for users

MARTIN TAYLOR

BLACKWELL SCIENTIFIC PUBLICATIONS
OXFORD LONDON EDINBURGH
BOSTON PALO ALTO MELBOURNE

Blackwell Scientific Publications
Editorial offices:
Osney Mead, Oxford, OX2 0EL
8 John Street, London, WC1N 2ES
23 Ainslie Place, Edinburgh, EH3 6AJ
52 Beacon Street, Boston
Massachusetts 02108, USA
667 Lytton Avenue, Palo Alto
California 94301, USA
107 Barry Street, Carlton,
Victoria 3053, Australia



First published 1985
Reprinted with corrections 1986

Phototypeset by
Oxford Computer Typesetting

Printed and bound in
Great Britain at
The Hollen Street Press Ltd

British Library
Cataloguing in Publication Data

Taylor, M.
Pick for users.
1. Pick (Computer operating system)
I. Title
001.64'25 QA76.76.063

ISBN 0-632-01492-X

DISTRIBUTORS

USA and Canada
Blackwell Scientific Publications Inc.
P O Box 50009, Palo Alto
California 94303

Australia
Blackwell Scientific Publications
(Australia) Pty Ltd
107 Barry Street,
Carlton, Victoria 3053

For Jillian

# Acknowledgement

The author would like to acknowledge the assistance given by Stuart Rees of Manchester Polytechnic in the preparation of this book.

# Contents

# Foreword by Pick Systems

Pick Systems has been labouring for many years to produce a computer operating system which is accessible by users. An operating system which enables users to utilise a computer without being buried in technical detail. Pick Systems believe that it has achieved this aim. It is a source of great satisfaction and a vindication of all of our efforts that this is being recognised. More computers are launched with the Pick operating system each year, the press follow our achievements and developments with increasing interest, conventions and exhibitions are convened which are devoted to the Pick operating system.

Pick is particularly pleased to see the publication of this book which, like the operating system, is aimed at the users of computers rather than technicians. Pick thinks that no business considering the purchase of a computer should fail to consider the Pick operating system and the information contained within these pages explains why.

**Frank Petyak**
*National Sales Manager*
*Pick Systems*

# Preface

This is a book about the facilities given to computers by the Pick™ Operating System[1]. Pick is a computer operating system which has been under development for over 20 years and has been commercially available since 1973. With the advent of the implementation of Pick on small multi-user microcomputers, Pick has seen an explosion of popularity. There are over 3500 applications written under Pick in just about every conceivable commercial requirement. There are 20 manufacturers whose computers have Pick implemented, or whose computer runs a variant of Pick, and there are more implementations being undertaken all the time. All software written under Pick may be transported across the whole of this range of equipment. If we include the Pick lookalikes in the Pick 'community' there should be over 60,000 computers running Pick and approaching 400,000 users of Pick throughout the world by the end of 1985.

The aim of this book is to impart practical information, of benefit to users and prospective users of Pick, rather than a comprehensive description of all Pick's facilities. A comprehensive description would take up much more space than one short book allows, and is in any case already available in the form of the Pick Reference Manual. This book's objective is to take the user to a point where further knowledge can easily be gleaned from the Reference Manual.

A particularly valuable feature of Pick is the ability to produce *ad hoc* reports using the enquiry language, Access. Consequently Access is described first, with examples from a simple personnel file. Access is certainly a tool that most departmental managers would wish to use and its use requires no previous technical knowledge.

In order to extend the reporting facilities for a Pick database and produce more sophisticated reports the user needs to know a little about the way that the database is structured and how the system editor works, so these are described next. The system editor is a utility which allows the modification of any item on the database. These sections will be of particular interest to people who are experienced in using Access and

1. PICK is a trademark of Pick Systems.

who wish to understand more about the capabilities of their Pick computer.

The next most important areas for users are the spooler (a utility which deals with the production of printed output) and the security system (which allows the computer manager to prevent access to sensitive data). These are described in detail, but an understanding of the security system can only follow a thorough understanding of the database, so a more detailed description of the database is given. This section of the book should provide enough detail for the computer manager to be able to cope with most day to day requirements.

The 'technical' aspects of Pick are Pick BASIC and the job control language, Proc, which links all the Pick features. The information given here describes concepts, rather than technical details, to enable the user to understand his application and, if necessary, to progress to a more technical level. Most users need to use Proc more than BASIC and tend to add to their Procs by changing menus and setting up permanent reports. To facilitate this, Proc is described in a tutorial fashion with the aim of showing the reader how Proc is used to set up menus and reports and to make the information easy to assimilate.

Pick 'lookalikes' have already been mentioned. These are computers with an operating system which looks like Pick, but which were not in fact implemented by, or in the same way as, Pick Systems. These tend to have some operational differences to the 'true' Pick standard, but since they can be viewed as part of the Pick 'community', these differences have been pointed out, where appropriate. The three main Pick lookalikes are McDonnell Douglas Reality/Sequoia, Prime Information and the personal computer database package, Revelation.

Finally, a word about the presentation of the book. It has been thought very important to give a practical approach throughout and many examples of actual operation have been given. To this end it has been necessary to indicate commands and responses with different typefaces. Commands typed into the computer are presented like this:

```
A COMMAND TYPED INTO THE COMPUTER BY THE USER
```

Where responses from the computer have been incorporated into the main text, the responses are in this typeface:

**A RESPONSE TO A COMMAND BY THE COMPUTER**

Other computer output, which might have appeared on a VDU screen or might have been hard copy on a printer, is presented in a photo-reduced format, as in Fig. P.1.

```
PAGE   1                                        15:50:53  04 DEC 1985

DEPARTMENT              RATE

PERSONNEL                4.00

PRODUCTION              13.38

TRANSPORT                8.50
                        25.88

6 ITEMS LISTED.
```

**Fig. P.1.**

# Chapter 1
# Introduction

Computers are devices which are becoming very familiar in our daily lives. Their uses are myriad and we can expect to encounter computers in many situations. Computers can be found in the home, in classrooms, in spacecraft and even in washing machines. Many computers are used by businesses for administrative purposes.

## HARDWARE AND SOFTWARE

When we talk about computers we usually envisage a box of electronic tricks consisting of wires, 'chips' and perhaps flashing lights, tapes and floppy disks. This is referred to as 'hardware'.

Nowadays most people are aware of the idea of a 'program' — some set of instructions that the computer can understand and interpret in a specific manner to carry out a task. This is referred to as 'software'.

When programming or simply using a computer, we sometimes take for granted the things which seem obvious, that we can 'talk' to a computer using a terminal or keyboard, and that our data will be stored in the memory of the computer or on a magnetic disk. These tasks, and many others, are carried out by special software called the 'operating system'.

This book is about one such operating system, called Pick. In fact the Pick Operating System carries out many tasks not normally associated with operating systems. Pick has its own 'database', which enables data to be stored and enquired upon simply. The enquiry facilities are given by means of a natural English (or French, Spanish or Japanese) language feature. Just as words in the English language might be defined in the Oxford English Dictionary, so words in the Pick enquiry language are also defined in a dictionary. This makes the database familiar to the user, apart from being open ended.

A special version of BASIC is used to update the information held in the database, this BASIC is designed around the database and interfaces with it very easily. Again the emphasis is very much on the data, rather than programming.

Computers running Pick are normally 'multi-user' computers. That is, more than one person may use the computer at one time. The various users of the computer might be carrying out completely different tasks. The fact that there are usually other users on the system is completely transparent. Pick provides facilities to help the user administer the problems caused by multi-user computers, such as providing queuing facilities for shared devices, like printers, in the form of a spooler. Pick also provides facilities to ensure that data can be kept secure, so that, for example, while the Chairman of the company may examine any data, the order entry clerk may be forbidden from looking at the personnel records.

Typically a computer system running Pick will consist of a central processing unit, where all the computing is actually carried out, and several terminals, which may be visual display units (VDUs), or teletype printers with a keyboard. It is through the terminals that we address the computer and access our data. All the information presented in this book relates to operations carried out at a terminal. From now on the computer hardware and the electronic tricks are irrelevant. It has no more importance than a filing cabinet in an office.

## LOGGING ON

When a terminal connected to a Pick computer is switched on a message appears on the screen:

**LOGON PLEASE:**

the computer is asking you to identify yourself and 'log on' to an 'account'. All Pick computers have an account called SYSPROG which is normally used only by the computer manager, but in principle the account name may be anything that has already been defined by the computer manager. Let us suppose that there is an account on our computer called ADMIN. We can log on to that account by typing ADMIN followed by the button marked RETURN or ENTER. Note that if the account is called ADMIN the computer will not accept 'admin'. Pick is 'case sensitive'. If a word is defined in upper case it may not also be used in lower case, unless of course it has also been defined in lower case.

If the logon name is rejected, the computer responds by complaining with the error message:

**USER ID?**

and then returns to the logon prompt.

**LOGON PLEASE:**

This is very characteristic of Pick. If you ever enter a command which Pick does not understand, there will be an error message. Usually this error message will help you to decide what was wrong and enter the correct command.

The next thing that might happen is that the computer will prompt with the word:

**PASSWORD:**

If this happens the account is protected by a password and we must know the password in order to proceed further. When the password is typed, the letters making up the password do not appear on the screen, this helps to preserve the security of sensitive data. If the password is typed incorrectly the computer will respond:

**PASSWORD?**

**LOGON PLEASE:**

and we have to start again from the account name.

Once this barrier has been overcome the computer will allow us to access the account. If we are accessing an account where a real application has already been set up, such as a personnel system or a sales ledger, it is more than likely that the master menu for the application will appear automatically. If there is no menu the computer will issue a 'prompt character' which on most systems will be a chevron (>) or it might be a colon (:). This is 'terminal command level' or TCL. The computer is ready for our next command and is asking us what we want to do next.

From TCL we may interrogate the database or execute a program or initiate any of the utilities that make up the Pick Operating System. One of the utilities available enables us to 'log off' the computer and return to the LOGON PLEASE: prompt. To log off we type the word OFF followed by the button marked RETURN or ENTER. This discipline of logging off should be carried out whenever we have finished using the computer.

# Chapter 2
# The ACCESS Enquiry Language

In normal circumstances the user of a computer will have an application running on the computer, with data and specific procedures already in place. His problem is then how to make the most effective use of his data outside the normal procedures. In this chapter we shall assume that there is an existing application with data stored on the Pick database. We shall see how the enquiry language may be used to answer *ad hoc* queries and how the data may be presented in new ways.

A database is a collection of data stored in an organised manner which enables the data to be accessed easily. A telephone directory is an example of a database which is not (yet!) found on a computer. The data may be as simple as a list of names and addresses or as complex as a sales ledger. In Pick, data is stored in 'files'. A file contains a collection of data which belongs in the same category. So, in business, one file might contain all the names and addresses of our customers, another might contain all the order details. Both files would be on the same database, so we would not duplicate the names and addresses of the customers in the orders on the order file.

The individual details of a single customer will be held together in a 'record'. Each record on the file will thus hold the name and address of a single customer. The name and the address are called 'fields'. There will be one special field of each record which distinguishes the records. This is called the 'key' or record identifying field. This has to be something unique amongst the records within the file. In the case of a customer list it might be an account number.

A computerised database has to have some method of retrieving the information held on the database. We may wish to view the data in many ways and interrogate the database. The aim is to answer questions like "how many customers have we in Kent?", or "which products do we sell the most of?". Pick provides a language to do this. The various distributors of Pick call this language by various names. ACCESS, RECALL, INFORM, R/LIST, ENGLISH, FRANÇAIS, ESPAGNOL

| Clock card number | Name | Address | Position | Department | Start date | Rate of pay | Telephone number | Sex | Age |
|---|---|---|---|---|---|---|---|---|---|
| A-100 | HALL F | 39 KING STREET<br>BRIGHTON<br>SUSSEX | SECRETARY | PERSONNEL | 11AUG83 | 4.00 | 790-2903 | M | 23 |
| B1-20 | JOHNSON D | 3 CARRBANK AV<br>HYDE<br>CHESHIRE | MANAGER | TRANSPORT | 01APR82 | 4.50 | 197-3582 | M | 26 |
| A-400 | THOMSON A J | 8 DUGDALE AV<br>CROWTHORNE<br>BERKS | CUTTER | PRODUCTION | 26JAN83 | 4.35 | 739-1095 | F | 31 |
| B-523 | WRIGHT J D | 4 PENDLEWAY<br>CAMBRIDGE | MACHINIST | PRODUCTION | 19FEB85 | 3.80 | 497-3528 | M | 31 |
| B1-1 | ELLIS K | 91 HOLLAND ST<br>ESHER<br>SURREY | CUTTER | PRODUCTION | 05MAR82 | 5.23 | - | F | 22 |
| C-10 | ROTHWELL T M | 230 HAMILTON ST<br>COVENTRY<br>WARWICKS | FITTER | TRANSPORT | 10JUL84 | 4.00 | 891-6867 | F | 50 |

**Fig. 2.1.** The data comprising the sample Personnel database.

and NIPPON-GO are examples of proprietary names for what is essentially the same language. The official Pick Systems trademark for the language is ACCESS so we shall call the enquiry language ACCESS throughout.

As the name ENGLISH implies, ACCESS is a language which is like everyday English, it is recognisable as English and is natural in use. To illustrate the examples in this section a small personnel file called PERSONNEL has been used. PERSONNEL contains the information shown in the table in Fig. 2.1 in six records. Note that in this example the 'special' or key field is the clock card number.

An Access enquiry sentence is typed at the keyboard and then sent to the computer by pressing the RETURN key. The command may be given at any point where you see the Terminal Command Level (TCL) prompt, which will be a chevron (>) or possibly a colon (:).

The first word of any Access sentence must be a verb. Examples of verbs are LIST, SORT, SELECT, SSELECT (Sort and select), COUNT and SUM. This tells the computer what to do.

The second word of an Access sentence is normally a file name. This tells the computer which data the operation is to be carried out on. The file name that will be used for all the examples in this chapter is called PERSONNEL.

Other words may follow the file name. The various words are separated by a space, just as in written English.

The simplest possible command is simply a verb and a file name, such as:

```
LIST PERSONNEL
```

which will give a display of all the records in the file, giving a preselected default report of certain fields from all the records. In our small sample file, this results in the display shown in Fig. 2.2. Just for now, take it for granted that a default report appears, the way that this is set up is described in the chapter on dictionaries.

## MODIFYING THE REPORT

The display above will appear on the terminal on which the command is typed. The report will halt at the end of the last line of each page and will only continue when the user presses the RETURN key. Instead of pressing RETURN the user might type CONTROL-X[1], at which point

1. CONTROL and X are two separate keys which have to be pressed together to generate the command, here called CONTROL-X. Do not try to press both keys simultaneously, but hold the CONTROL key down and type X.

```
PAGE  1                                  09:02:35  12 DEC 1985

PERSONNEL  NAME           POSITION     RATE    STARTED

A-100      HALL F         SECRETARY    4.00 11 AUG 1983
B1-20      JOHNSON D      MANAGER      4.50 01 APR 1982
A-400      THOMSON A J    CUTTER       4.35 26 JAN 1983
B1-1       ELLIS K        CUTTER       5.23 05 MAR 1982
B-523      WRIGHT J D     MACHINIST    3.80 19 FEB 1985
C-10       ROTHWELL T M   FITTER       4.00 10 JUL 1984

6 ITEMS LISTED.
```

**Fig. 2.2.** The output produced by LIST PERSONNEL

the report will be terminated and control returned to TCL. This form of display can be changed by adding an 'output modifier' to the Access sentence. For example, the word LPTR (short for lineprinter) will cause the output to appear on a printer, rather than the user's terminal. The option (P) may be used instead; this has the same meaning as LPTR. Options are single character codes surrounded by brackets. More than

**MODIFIERS**

| | |
|---|---|
| LPTR | Send output to the system printer. |
| NOPAGE | Do not halt at the end of the last line of each page. |
| HDR-SUPP | Do not display the page heading. |
| ID-SUPP | Do not display the key field. |
| ONLY | Only display the key field (i.e. instead of the default report). |
| DBL-SPC | Leave a blank line between each line output. |
| COL-HDR-SUPP | Do not display the column headings. |
| HEADING | Use a different heading. |
| FOOTING | Use a footing. |
| TOTAL | Total a numeric field. |
| DET-SUPP | Display only TOTAL lines. |

**OPTIONS**

| | |
|---|---|
| C | Same as COL-HDR-SUPP |
| D | Same as DET-SUPP |
| H | Same as HDR-SUPP |
| I | Same as ID-SUPP |
| N | Same as NOPAGE |
| P | Same as LPTR |

one option may be specified by separating the single character codes by commas.

These options and modifiers may be included in the Access sentences as required, e.g.

```
LIST PERSONNEL ID-SUPP DBL-SPC NOPAGE

LIST PERSONNEL DBL-SPC (I,N)
```

Either of these sentence will produce the report shown in Fig. 2.3.

```
PAGE   1                              09:04:42  12 DEC 1985

NAME           POSITION      RATE    STARTED

HALL F         SECRETARY     4.00 11 AUG 1983

JOHNSON D      MANAGER       4.50 01 APR 1982

THOMSON A J    CUTTER        4.35 26 JAN 1983

ELLIS K        CUTTER        5.23 05 MAR 1982

WRIGHT J D     MACHINIST     3.80 19 FEB 1985

ROTHWELL T M   FITTER        4.00 10 JUL 1984


6 ITEMS LISTED.
```

**Fig. 2.3.** The output produced by
LIST PERSONNEL DBL-SPC (I,N)

In Fig. 2.3 the clock card number has been omitted from the display and each line of the report has been separated by a blank line. The report is said to be 'double spaced'.

Note that, except on McDonnell Douglas systems, options must be placed at the end of the sentence otherwise they will be ignored.

## SELECTING FIELDS TO BE DISPLAYED

The information which appears on this 'default' report is not the only way of displaying information held on the file. If a list of field names is added to the Access sentence, then the information stored in those fields will be displayed instead of the default report. Any number of fields may be requested in an Access sentence. The field names may appear anywhere in the command after the file name and in any order. The fields will be displayed as separate columns if the terminal is wide

enough, otherwise each field will occupy a separate line. The column displayed on the left will be the 'special' key field but this may be suppressed with the ID-SUPP modifier. Subsequent columns will show the information from each of the fields specified. These will appear in the same order in which they were requested.

Figures 2.4, 2.5 and 2.6 show three examples of Access commands and reports where the default report has been suppressed and replaced by particular fields. In Fig. 2.4:

```
LIST PERSONNEL NAME ADDRESS
```

only the name and address of each employee are shown. Note how Pick seems to know how the address should be formatted, with each line of the address on a separate line on the report.

The next example, Fig. 2.5,

```
LIST PERSONNEL NAME RATE POSITION
```

shows the name, the rate of pay and the position of each of the employees. In this report note how Pick seems to know that textual data, like the name and the position, are formatted one under another starting from the left, but numerical data, like the rate of pay, are formatted as we expect to see columns of numbers, formatted to the right.

The Access sentence:

```
LIST PERSONNEL SURNAME OCCUPATION NUMBER (I)
```

shown in Fig. 2.6 shows how we can define words which actually manipulate the data stored on the file, as in SURNAME, and define words which mean the same as other words, synonyms like OCCUPATION and NUMBER. It also shows how we may combine the ideas of Access together. Here we have asked that the key field should not be displayed by adding the I option. The way in which these words are defined is explained in the chapter on dictionaries.

## USING ACCESS TO ASK QUESTIONS

More often than not we wish to ask questions of the database, rather than simply specifying the information to be displayed. These questions may take the form of: "Which of our secretaries speak French?"; "Which customers with a credit limit over a thousand pounds pay late?" and so on.

```
PAGE   1                              09:09:08  12 DEC 1985

PERSONNEL  NAME          ADDRESS

A-100      HALL F        39 KING STREET
                         BRIGHTON
                         SUSSEX
B1-20      JOHNSON D     3 CARRBANK AV
                         HYDE
                         CHESHIRE
A-400      THOMSON A J   8 DUGDALE AV
                         CROWTHORNE
                         BERKS
B1-1       ELLIS K       91 HOLLAND ST
                         ESHER
                         SURREY
B-523      WRIGHT J D    4 PENDLEWAY
                         CAMBRIDGE
C-10       ROTHWELL T M  230 HAMILTON ST
                         COVENTRY
                         WARWICKS

6 ITEMS LISTED.
```

**Fig. 2.4.** The output produced by
LIST PERSONNEL NAME ADDRESS

```
PAGE   1                              09:10:30  12 DEC 1985

PERSONNEL  NAME          RATE  POSITION

A-100      HALL F        4.00  SECRETARY
B1-20      JOHNSON D     4.50  MANAGER
A-400      THOMSON A J   4.35  CUTTER
B1-1       ELLIS K       5.23  CUTTER
B-523      WRIGHT J D    3.80  MACHINIST
C-10       ROTHWELL T M  4.00  FITTER

6 ITEMS LISTED.
```

**Fig. 2.5.** The output produced by
LIST PERSONNEL NAME RATE POSITION

```
PAGE   1                                  09:11:49  12 DEC 1985

SURNAME    OCCUPATION    NUMBER

HALL       SECRETARY      A-100
JOHNSON    MANAGER        B1-20
THOMSON    CUTTER         A-400
ELLIS      CUTTER          B1-1
WRIGHT     MACHINIST      B-523
ROTHWELL   FITTER          C-10

6 ITEMS LISTED.
```

**Fig. 2.6.** The output produced by
LIST PERSONNEL SURNAME OCCUPATION NUMBER (I)

We do this by adding the word WITH (or IF which means the same thing) followed by the name of the field and relational operator, followed by the value to be compared enclosed in double quotes. Relational operators are simply words or abbreviations or symbols which specify the type of comparison to be carried out.

| Relational operator | Meaning |
|---|---|
| EQ or = or null | Equal to |
| GT or AFTER or > | Greater than |
| LT or BEFORE or < | Less than |
| GE or >= | Greater than or equal to |
| LE or <= | Less than or equal to |
| NE or NOT or # | Not equal to |
| NO | Having no value. |

Here are a few examples:

```
LIST PERSONNEL WITH POSITION "SECRETARY"
LIST PERSONNEL IF POSITION = "SECRETARY"
LIST PERSONNEL IF AGE GE "25"
LIST PERSONNEL WITH NAME < "M"
LIST PERSONNEL WITH STARTED AFTER "31 JUL 1983"
LIST PERSONNEL WITH DEPARTMENT NE "TRANSPORT"
LIST PERSONNEL WITH NO PHONE
LIST PERSONNEL NAME ADDRESS WITH NO PHONE (I)
```

Figure 2.7 shows those members of staff who joined the company since 31 July 1983 and is the result of the Access sentence:

```
LIST PERSONNEL WITH STARTED AFTER "31 JUL 1983"

PAGE   1                                  09:17:27  12 DEC 1985

PERSONNEL NAME           POSITION      RATE     STARTED

A-100      HALL F        SECRETARY     4.00 11 AUG 1983
B-523      WRIGHT J D    MACHINIST     3.80 19 FEB 1985
C-10       ROTHWELL T M  FITTER        4.00 10 JUL 1984

3 ITEMS LISTED.
```

**Fig. 2.7.** The output produced by
LIST PERSONNEL
WITH STARTED AFTER "31 JUL 1983"

These criteria may be joined together to modify the restrictions by the use of the connectives AND and OR.

For example, Fig. 2.8 shows the report produced by the sentence:

LIST PERSONNEL WITH POSITION "CUTTER" AND WITH RATE < "5"

and Fig. 2.9 shows the report produced when the command

LIST PERSONNEL WITH POSITION "CUTTER" OR WITH POSITION "SECRETARY"

is given.

```
PAGE   1                                  09:25:19  12 DEC 1985

PERSONNEL NAME           POSITION      RATE     STARTED

A-400      THOMSON A J   CUTTER        4.35 26 JAN 1983

END OF LIST
```

**Fig. 2.8.** The output produced by
LIST PERSONNEL WITH POSITION "CUTTER"
AND WITH RATE < "5"

The up arrow character may be used as a 'wild card' character, i.e. the sentence

LIST PERSONNEL WITH RATE "4.^^"

will display all those employees whose RATE field contains a four

```
PAGE   1                                09:29:20  12 DEC 1985

PERSONNEL NAME          POSITION      RATE    STARTED

A-100      HALL F       SECRETARY     4.00 11 AUG 1983
A-400      THOMSON A J  CUTTER        4.35 26 JAN 1983
B1-1       ELLIS K      CUTTER        5.23 05 MAR 1982

3 ITEMS LISTED.
```

**Fig. 2.9.** The output produced by
LIST PERSONNEL WITH POSITION "CUTTER"
OR WITH POSITION "SECRETARY"

character field beginning with 4., so Fig. 2.10 shows everyone who earns between 4.00 and 4.99 pounds an hour.

```
PAGE   1                                09:34:05  12 DEC 1985

PERSONNEL NAME          POSITION      RATE    STARTED

A-100      HALL F       SECRETARY     4.00 11 AUG 1983
B1-20      JOHNSON D    MANAGER       4.50 01 APR 1982
A-400      THOMSON A J  CUTTER        4.35 26 JAN 1983
C-10       ROTHWELL T M FITTER        4.00 10 JUL 1984

4 ITEMS LISTED.
```

**Fig. 2.10.** The output produced by
LIST PERSONNEL WITH RATE "4.^^^"

Access can also search for patterns of characters by including square brackets within the double quotes.

For example, the Access sentence:

```
LIST PERSONNEL WITH PHONE "790]"
```

will display all the staff whose telephone number begins with the code 790, as shown in Fig. 2.11. This also shows that the field which is the subject of our enquiry need not necessarily be shown in our report.

```
PAGE   1                                09:35:59  12 DEC 1985

PERSONNEL NAME          POSITION      RATE    STARTED

A-100      HALL F       SECRETARY     4.00 11 AUG 1983

END OF LIST
```

**Fig. 2.11.** The output produced by
LIST PERSONNEL WITH PHONE "790]"

In the same way:

```
LIST PERSONNEL WITH STARTED = "[1983"
```

will display only the PERSONNEL who joined the company in 1983.

We can specify that a field must contain a sequence of characters by surrounding the text with square brackets. Figure 2.12 shows any PERSONNEL whose POSITION contains the characters IN and is the result of the sentence:

```
LIST PERSONNEL WITH POSITION "[IN]"
```

```
PAGE   1                              09:43:21  12 DEC 1985

PERSONNEL NAME           POSITION      RATE    STARTED

B-523     WRIGHT J D     MACHINIST     3.80 19 FEB 1985

END OF LIST
```

**Fig. 2.12.** The output produced by
LIST PERSONNEL WITH POSITION "[IN]"

Record keys may be included in an Access sentence to specify that only those indicated should be displayed. Single quotes must surround the keys. Figure 2.13 shows the report produced from the Access sentence:

```
LIST PERSONNEL 'B-523' 'C-10'
```

This displays only those employees with the specified record keys.

```
PAGE   1                              09:44:47  12 DEC 1985

PERSONNEL NAME           POSITION      RATE    STARTED

B-523     WRIGHT J D     MACHINIST     3.80 19 FEB 1985
C-10      ROTHWELL T M   FITTER        4.00 10 JUL 1984

2 ITEMS LISTED.
```

**Fig. 2.13.** The output produced by
LIST PERSONNEL 'B-523' 'C-10'

## SORTING

Usually we wish to SORT reports in some sequence so that the information is readily accessible. It would be extremely difficult to find some-

one's telephone number if the telephone directory were not arranged alphabetically. In fact Pick provides sorting facilities through the SORT verb. The verb SORT gives the same output as LIST but sorted. The sentence

```
SORT PERSONNEL
```

will give the same output as LIST PERSONNEL but the records will be sorted in the ascending order of the key field, which in this case is the clock card number, giving the report shown in Fig. 2.14.

```
PAGE   1                              09:45:54  12 DEC 1985

PERSONNEL NAME           POSITION      RATE     STARTED

A-100     HALL F         SECRETARY     4.00 11 AUG 1983
A-400     THOMSON A J    CUTTER        4.35 26 JAN 1983
B-523     WRIGHT J D     MACHINIST     3.80 19 FEB 1985
B1-1      ELLIS K        CUTTER        5.23 05 MAR 1982
B1-20     JOHNSON D      MANAGER       4.50 01 APR 1982
C-10      ROTHWELL T M   FITTER        4.00 10 JUL 1984

6 ITEMS LISTED.
```

**Fig. 2.14.** The output produced by SORT PERSONNEL

Any other field may be sorted by adding the word BY and the name of the field to be sorted.

```
SORT PERSONNEL BY NAME
```

will give the same display as Fig. 2.14 sorted alphabetically by NAME, this is the report shown in Fig. 2.15.

```
PAGE   1                              09:47:58  12 DEC 1985

PERSONNEL NAME           POSITION      RATE     STARTED

B1-1      ELLIS K        CUTTER        5.23 05 MAR 1982
A-100     HALL F         SECRETARY     4.00 11 AUG 1983
B1-20     JOHNSON D      MANAGER       4.50 01 APR 1982
C-10      ROTHWELL T M   FITTER        4.00 10 JUL 1984
A-400     THOMSON A J    CUTTER        4.35 26 JAN 1983
B-523     WRIGHT J D     MACHINIST     3.80 19 FEB 1985

6 ITEMS LISTED.
```

**Fig. 2.15.** The output produced by SORT PERSONNEL BY NAME

Note that if you are suppressing the default report, and wish the sorted field to be displayed, the name of the sorted field must be specified in the list of fields to be displayed, as well as in the sort criteria. Figure 2.16 shows the report produced by:

```
SORT     PERSONNEL      BY NAME      NAME STARTED AGE
(verb)   (file)         (sort by)    (display fields)
```

```
PAGE   1                              09:49:21  12 DEC 1985

PERSONNEL NAME              STARTED    AGE

B1-1      ELLIS K       05 MAR 1982   22
A-100     HALL F        11 AUG 1983   23
B1-20     JOHNSON D     01 APR 1982   26
C-10      ROTHWELL T M  10 JUL 1984   50
A-400     THOMSON A J   26 JAN 1983   31
B-523     WRIGHT J D    19 FEB 1985   31

6 ITEMS LISTED.
```

**Fig. 2.16.** The output produced by
SORT PERSONNEL BY NAME
NAME STARTED AGE

The sorting sequence may be reversed by using the word BY-DSND (by descending) instead of BY, as in Fig. 2.17.

```
SORT PERSONNEL BY-DSND NAME NAME STARTED AGE
```

```
PAGE   1                              09:52:07  12 DEC 1985

PERSONNEL NAME              STARTED    AGE

B-523     WRIGHT J D    19 FEB 1985   31
A-400     THOMSON A J   26 JAN 1983   31
C-10      ROTHWELL T M  10 JUL 1984   50
B1-20     JOHNSON D     01 APR 1982   26
A-100     HALL F        11 AUG 1983   23
B1-1      ELLIS K       05 MAR 1982   22

6 ITEMS LISTED.
```

**Fig. 2.17.** The output produced by
SORT PERSONNEL BY-DSND NAME
NAME STARTED AGE

The most significant sort key is specified first so that

```
SORT PERSONNEL BY DEPARTMENT BY AGE NAME DEPARTMENT AGE
```

which is shown in Fig. 2.18, will give a different output to

```
SORT PERSONNEL BY AGE BY DEPARTMENT NAME DEPARTMENT AGE
```

which is shown in Fig. 2.19.

```
PAGE   1                              09:53:24  12 DEC 1985

PERSONNEL NAME          DEPARTMENT            AGE

A-100     HALL F        PERSONNEL              23
B1-1      ELLIS K       PRODUCTION             22
A-400     THOMSON A J   PRODUCTION             31
B-523     WRIGHT J D    PRODUCTION             31
B1-20     JOHNSON D     TRANSPORT              26
C-10      ROTHWELL T M  TRANSPORT              50

6 ITEMS LISTED.
```

**Fig. 2.18.** The output produced by
SORT PERSONNEL BY DEPARTMENT BY AGE
NAME DEPARTMENT AGE

```
PAGE   1                              09:55:00  12 DEC 1985

PERSONNEL NAME          DEPARTMENT            AGE

B1-1      ELLIS K       PRODUCTION             22
A-100     HALL F        PERSONNEL              23
B1-20     JOHNSON D     TRANSPORT              26
A-400     THOMSON A J   PRODUCTION             31
B-523     WRIGHT J D    PRODUCTION             31
C-10      ROTHWELL T M  TRANSPORT              50

6 ITEMS LISTED.
```

**Fig. 2.19.** The output produced by
SORT PERSONNEL BY AGE BY DEPARTMENT
NAME DEPARTMENT AGE

## HEADINGS AND FOOTINGS

The output modifiers HEADING and FOOTING are different from the other modifiers in that they are not sufficient on their own to complete the task in hand. A value, surrounded by double quotes, must immediately follow HEADING or FOOTING. The text which is to appear as the heading or footing is placed inside the double quotes. In addition a number of output control options may appear within the double quote

surrounded by single quotes. These options are:

| | |
|---|---|
| B | Insert the curent break value (see BREAK-ON below) |
| C | Centre the current line |
| D | Insert the current date |
| F | Insert the current file name |
| L | Start a new line |
| P | Insert the current page number |
| T | Insert the current time and date |
| " | Print a single quote in the HEADING |

For example, Fig. 2.20 shows a report with a centred heading saying 'Personnel information' and the page number followed by two blank lines, produced by the command:

```
LIST PERSONNEL HEADING "Personnel information 'C' Page 'PLL'"
```

```
            Personnel information  Page    1


PERSONNEL NAME           POSITION     RATE     STARTED

A-100     HALL F         SECRETARY    4.00 11 AUG 1983
B1-20     JOHNSON D      MANAGER      4.50 01 APR 1982
A-400     THOMSON A J    CUTTER       4.35 26 JAN 1983
B1-1      ELLIS K        CUTTER       5.23 05 MAR 1982
B-523     WRIGHT J D     MACHINIST    3.80 19 FEB 1985
C-10      ROTHWELL T M   FITTER       4.00 10 JUL 1984
```

**Fig. 2.20.** The output produced by
LIST PERSONNEL
HEADING "Personnel information 'C' Page 'PLL'"

## BREAKING UP THE DATA INTO SECTIONS

A listing can be sectioned by using the BREAK-ON modifier. The end of a section in the report can be indicated by the value of one or more fields of information changing their value. For example, if we wished to separate the data relating to men from that relating to women, the point at which the SEX field became M would indicate the end of the women's section. BREAK-ON in front of a field name will cause Access to detect the changing field values and section the report. It is normally only employed in conjunction with a SORT type verb. If BREAK-ON is used with LIST a break would occur every time two records, physically

next to each other in the file, had different values in the field being broken on.

For example, to separate the employees into male and female, the sentence:

```
SORT PERSONNEL BY SEX BY NAME NAME AGE BREAK-ON SEX
```

would be used; this is shown in Fig. 2.21.

```
PAGE   1                              09:59:33  12 DEC 1985

PERSONNEL NAME          AGE SEX

B1-1      ELLIS K        22 F
C-10      ROTHWELL T M   50 F
A-400     THOMSON A J    31 F

                            ***

A-100     HALL F         23 M
B1-20     JOHNSON D      26 M
B-523     WRIGHT J D     31 M

                            ***
***

6 ITEMS LISTED.
```

**Fig. 2.21.** The output produced by
SORT PERSONNEL BY SEX BY NAME
NAME AGE BREAK-ON SEX

The three asterisks, ***, that are used to break up the report indicate the column causing the break. *** is the default used by BREAK-ON and may be changed, as you will see.

Any TOTAL fields will have the TOTAL for that section of the report displayed on the BREAK-ON line. For example:

```
SORT PERSONNEL BY DEPARTMENT BREAK-ON DEPARTMENT NAME TOTAL
RATE
```

produces the output shown in Fig. 2.22.

## SUMMARY REPORTS

By using SORT in conjunction with BREAK-ON, TOTAL and DET-SUPP it is possible to produce summary reports which only display total lines. This technique is very useful when answering questions like, "What are the total sales from each salesman?", where the detail of each

```
PAGE    1                              10:01:26  12 DEC 1985

PERSONNEL  DEPARTMENT          NAME          RATE

A-100      PERSONNEL           HALL F         4.00

           ***                                4.00

A-400      PRODUCTION          THOMSON A J    4.35
B-523      PRODUCTION          WRIGHT J D     3.80
B1-1       PRODUCTION          ELLIS K        5.23

           ***                               13.38

B1-20      TRANSPORT           JOHNSON D      4.50
C-10       TRANSPORT           ROTHWELL T M   4.00

           ***                                8.50

***                                          25.88

6 ITEMS LISTED.
```

**Fig. 2.22.** The output produced by
SORT PERSONNEL BY DEPARTMENT
BREAK-ON DEPARTMENT NAME TOTAL RATE

sale made by the salesmen is not required. An example from the PERSONNEL file will show the total rates of pay of the personnel working in each department:

```
SORT PERSONNEL BY DEPARTMENT BREAK-ON DEPARTMENT TOTAL RATE
DET-SUPP ID-SUPP
```

This is shown in Fig. 2.23.

```
PAGE    1                              10:03:09  12 DEC 1985

DEPARTMENT          RATE

PERSONNEL            4.00

PRODUCTION          13.38

TRANSPORT            8.50
                    25.88

6 ITEMS LISTED.
```

**Fig. 2.23.** The output produced by
SORT PERSONNEL BY DEPARTMENT
BREAK-ON DEPARTMENT TOTAL RATE
DET-SUPP ID-SUPP

In this case the *** from the total lines in the earlier report are replaced by the department names because none of the details of the report is being shown.

The BREAK-ON field name may be followed by text in double quotes. The text will be printed on each break line instead of the *** which is printed as a default. The following options may be included within the text surrounded by single quotes in the same way as the HEADING options discussed above.

| | |
|---|---|
| B | Specifies that this field is the field to be used in place of the HEADING B option. |
| D | Do not display the BREAK if there has only been one item of data since the last BREAK. |
| L | Stops a blank line being output before the BREAK. |
| N | Resets the page number to one after the BREAK. |
| P | Start a new page after the BREAK. |
| U | Underline any fields which are TOTAL fields. |
| V | Insert the value of the BREAK field at this point in the text. |

Figure 2.24 shows how the report can be broken up with a descriptive piece of text instead of the ***, and is the report produced by:

```
SORT PERSONNEL BY DEPARTMENT BREAK-ON DEPARTMENT "Total rates of
pay in 'V'" NAME TOTAL RATE
```

## OTHER ACCESS VERBS

Although SORT and LIST are probably the most commonly used Access verbs, this section would not be complete without a brief overview of the capabilities of the other verbs.

SELECT and SSELECT do not produce any output, apart from the number of records selected. SELECT is used to pass the output of an Access sentence into another process. The second process could be a BASIC program or another Access sentence, for instance. In its simplest form SELECT will pass a list of record keys into the secondary process. The secondary process can then read the records and process them in whatever way is required. This is typical of the way that one would produce reports on pre-printed stationery. SSELECT is the same as SELECT except that the records will be sorted into some sequence. This means that it should never be necessary to write a sort routine in BASIC and that all sorts can be done at the operating system level.

```
PAGE   1                              10:04:59  12 DEC 1985

PERSONNEL DEPARTMENT            NAME          RATE

A-100     PERSONNEL             HALL F         4.00

          Total rates of pay in PERSONNEL      4.00

A-400     PRODUCTION            THOMSON A J    4.35
B-523     PRODUCTION            WRIGHT J D     3.80
B1-1      PRODUCTION            ELLIS K        5.23

          Total rates of pay in PRODUCTION    13.38

B1-20     TRANSPORT             JOHNSON D      4.50
C-10      TRANSPORT             ROTHWELL T M   4.00

          Total rates of pay in TRANSPORT      8.50

***                                           25.88

6 ITEMS LISTED.
```

**Fig. 2.24.** The output produced by
SORT PERSONNEL BY DEPARTMENT
BREAK-ON DEPARTMENT "Total rates of pay in 'V'"
NAME TOTAL RATE

By extending SELECT with a list of display fields, as in the command:

```
SELECT PERSONNEL WITH AGE >= "25" NAME STARTED
```

the list of record keys is replaced by the values of the display fields. In this case, the name and the starting date would be selected instead of the clock card number. This has two very useful effects. Firstly, this can be used as mail merge data with RUNOFF. Secondly, this data can be passed in the normal manner into a BASIC program. Thus it will not be necessary for the BASIC program to access the file, retrieve the records and sort out the data required. All the retrieval is carried out in the SELECT pass through the file with a consequential increase in efficiency.

COUNT returns the number of records in the file which match whatever selection criteria are specified. Thus COUNT PERSONNEL results in the output "6 ITEMS COUNTED". SUM is used to total a given field. SUM PERSONNEL RATE would return "TOTAL OF RATE IS 25.88".

T-DUMP and S-DUMP direct the output of the Access sentence to the currently active tape or floppy disc backup device. If no display

fields are specified, the whole record is dumped to the tape. If display fields are specified then only those fields will be dumped.

T-LOAD has the reverse effect of T-DUMP, reading data from the tape and storing it on the file indicated. If any selection criteria are specified, only the records which obey the selection criteria will be retrieved from the tape. Both T-DUMP and T-LOAD are discussed in more detail in the chapter on archiving.

LIST-LABEL and SORT-LABEL output the results of the Access sentence to the screen or printer, but instead of being in a columnar format the data is rearranged so that it is suitable for printing on sticky labels. When the LIST-LABEL command has been typed, the system prompts for a second set of details which are used to determine how many labels are to be printed across the page, the distance between them, the height and width of the labels and the indentations. These parameters enable the LIST-LABEL and SORT-LABEL verbs to be totally general purpose and capable of printing any label format.

REFORMAT and SREFORMAT direct the output of an Access sentence to another file. Each field specified as a display field will be regarded as a sequential description of the record structure of the destination file, the first field being used as a record key. When used to their best effect these verbs are very useful for creating analysis files with information derived from master files and transaction files.

# Chapter 3
# Introduction to the Pick Database

In the last chapter we assumed that our application and data already existed and saw how we might formulate enquiries using data from a specific file. In this chapter we will see how the database is arranged, how accounts and files are created and how records are structured.

## THE ARRANGEMENT OF THE DATABASE

The Pick database is arranged in a three level hierarchy.

At the highest level the SYSTEM consists of a number of ACCOUNTS, or users. Each account may access any number of FILES, or it may share data by accessing files in other accounts.

Files are split into two portions, a dictionary portion, and a data portion. The dictionary contains records which define the structure of the records in the data portion. We usually refer to these records as ITEMS. Any file may contain any number of items. Pick will automatically allocate the required disk space and the file will grow or shrink as needed. The file size is limited only by the amount of disk space available.

This hierarchy of files can be represented diagrammatically (Fig. 3.1).

Access to the records within any file may be made randomly. Records consist of an item identifying field, the ITEM-ID, and then any number of fields for the data. The item-id is used as a reference to the record, as such every record in the file must have a unique item-id. The item-id may be regarded as the 'name' of the record or its 'label'. We shall often refer to the item-id as the 'key' field. Usually we choose some unique aspect of the data to be the item-id. For a customer file the item-id might be the customer's account number, for the personnel file discussed in the last chapter we chose the clock card number, or we might have chosen the national insurance number. If there is no unique aspect to the data we might allocate a sequential number as the item-id of each record.

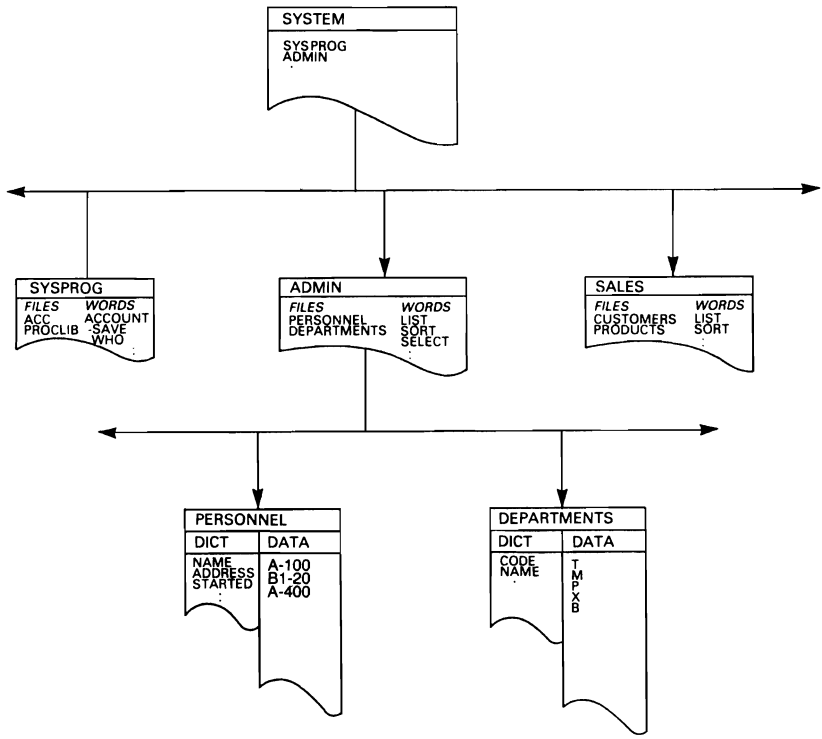



**Fig. 3.1.** The Pick File Hierarchy.

Any field may be broken up into two further sub-divisions so that, for instance, an address would take up only one field with the different lines of the address occupying subfields.

We call the fields ATTRIBUTES, the first sub-divisions VALUES, and the second sub-divisions SUB-VALUES.

The amount of disk space used is minimised, each record occupying exactly as much disk space as is required by the data. Files may contain records with 1 field of data alongside records with 1000 fields of data. Only as much disk space as is actually required will be taken up. Records are of completely variable length up to a maximum item size of 32,767 characters (32 Kbytes).

Each record in the data portion of the customer file would be of this structure. The dictionary of the customer file would contain records called ACCOUNT.NUMBER, NAME, ADDRESS and so on. This enables Access to format reports with the correct data. However, Access cannot be used to create the database in the first place.

Typical record structure for a customer

| Field number | Description |
|---|---|
| Item-id | Customer account number |
| 1 | Name |
| 2 | Address line 1, line 2, line 3 etc. |
| 3 | Telephone number |
| 4 | Credit limit |
| 5 | Contact name |
| etc. | |

Most businesses will purchase application programs, written in Pick BASIC, to update the database. A single application may be spread over several accounts. For a fully integrated accounting system, there may be one account for the users dealing with order entry, another account for the sales ledger, another for the purchase ledger and so on. The datafiles being used by the various aspects of the application would probably be shared.

When a user turns on a terminal attached to a Pick system, he will be confronted by a LOGON prompt and he is invited to type in the name of the account that he wishes to access. In effect he is entering the system via the system file or system dictionary. This is the highest level file on the system, and through typing an account name which tallies with one of the entries on the system dictionary, he logs on to the appropriate account and obtains the privileges associated with it. His presence on the account is logged in the ACC file and may be monitored by the computer manager using the LISTU utility.

Every Pick computer has an account called SYSPROG. This is usually only used by the person responsible for the management of the use of the computer. From now on we shall refer to this person as the 'system administrator'. If we logon to SYSPROG, we are able to execute many commands which are not available in any other account. Notably we may initiate archiving routines.

## CREATING ACCOUNTS

It is from SYSPROG that the creation of the database begins. In the last chapter we saw reference made to a file called PERSONNEL. In the introduction we assumed that an account called ADMIN was present on the computer that we could log to. We say that the file PERSONNEL is in the account ADMIN.

But accounts and files have to be created before they can be used. Accounts are created by using a 'verb' or utility, available for use only in the SYSPROG account, called CREATE-ACCOUNT. To create a new account, the command CREATE-ACCOUNT is entered at TCL:

```
CREATE-ACCOUNT
```

This initiates a series of prompts that are used to set up the account. The following example of replies to these prompts represents the simplest set of responses. <CR> means press RETURN without typing in anything else. This sets up the default values which are indicated. Anything else which is typed in before the carriage return would overwrite the default values:

```
ACCOUNT NAME? ADMIN
L/RET CODES? <CR>  (default — no codes)
L/UPD CODES? <CR>  (default — no codes)
PRIVILEGES? <CR>   (default — minimum privileges)
MOD,SEP? <CR>      (default — 29,1)

[417] FILE 'ADMIN' CREATED: BASE = 29221, MODULO = 29, SEPAR = 1

273 ITEMS COPIED
'ADMIN' ADDED
'ADMIN' UPDATED

PASSWORD? <CR>     (default — no password required for this
                    account)
```

The prompts deal with the security and size of the account. These subjects are dealt with in the chapter on security. The creation of the account and the copying of records into it enable the account to be logged to and give the basic vocabulary, in terms of commands, to the account. As yet, there are no data files within the account, except one, the "Master Dictionary" or MD, in which the basic vocabulary is now defined.

## CREATING FILES

We can now log to the new account by entering LOGTO ADMIN. There is no password, so the computer will transfer us directly to

ADMIN. There is, as yet, no logon process for the account, so the computer will prompt at TCL.

To create a file the command CREATE-FILE is used; this command would create a file called PERSONNEL (provided the command is given in an account with sufficient privileges):

```
CREATE-FILE PERSONNEL 3 23
```

```
[417] FILE 'PERSONNEL' CREATED; BASE = 32089, MODULO = 3,
SEPAR = 1
```

```
[417] FILE 'PERSONNEL' CREATED: BASE = 32224, MODULO = 23,
SEPAR = 1
```

The numbers in the command represent the initial size of the file. The significance of the size of a file is discussed in Chapter 7. The computer appears to have created two files. In fact it has created a dictionary, to hold records with names such as ADDRESS, POSITION, STARTED and so on, as well as a file which will actually hold the data for the PERSONNEL file.

## FILES AND RECORDS

Each data file that may be accessed has an associated data dictionary which describes the data that is held within it. This data dictionary is used by Access to extend the user's vocabulary when referencing any particular file. The dictionary may also contain descriptions of data which is held physically on other files but referenced by some key information held on the first file. This is often referred to as a JOIN.

At TCL the utilities LISTFILES, LISTPROCS, LISTVERBS and LISTDICT may be used to determine what vocabulary is open to any particular user.

Every file and dictionary on the system is of the same physical format, right down from the system dictionary — there are no special forms. However some files may be "single level" — that is, they are dictionary and data combined. The system dictionary and account master dictionaries are examples of these. Furthermore, each file or dictionary consists of records. There is no limit to the number of records on any file.

From this it follows that since file definitions are merely records in the master dictionary, there is no limit to the number of files that may be accessed from any particular account. Records consist of a record key followed by a number of fields, known as attributes.

| | |
|---|---|
| B-523 | ..... Record key or item-id |
| 001 WRIGHT J D | ..... Attribute 1 |
| 002 4 PENDLEWAY]CAMBRIDGE | ..... Attribute 2 (multi-values) |
| 003 MACHINIST | ..... Attribute 3 |
| 004 6129 | etc. |
| 005 380 | |
| 006 497-3528 | |
| 007 M | |
| 008 31 | |

The physical format of one of the PERSONNEL records.

Any record may be of any length up to 32 Kbytes. Within this it may consist of any number of attributes. Unused attributes occupy 1 byte unless they are at the end of the record in which case they occupy no space. Attributes may be of any size, up to the record limit and the size need not be predetermined in any file definition. Indeed corresponding attributes in different records can be of completely different lengths without taking up any more disk space than is actually necessary for the data.

Records may just consist of attributes, but the Pick System allows attributes to be subdivided into two further logical levels, termed multi-values and sub-values. These would normally be used for repeating groups of data, such as lines on a customer order. Some programmers will find it convenient to use multi-values when writing an application, because instead of having to have one record per line, the whole order may be contained in one record, and yet there will be no loss in flexibility as the number of multi-values is not predetermined. A very strong argument can be made for using multi-values for data such as addresses, where an address may be defined as one multi-valued field and would thus have an indeterminate number of lines, whereas on conventional database systems the number of lines on addresses has to be predetermined, with consequent loss of flexibility.

# Chapter 4
# The System Editor

The system editor is a utility which permits on-line modification of any item in the database. It may be used to create BASIC programs, Procs, data records or dictionaries. It is a line editor — that is, at any one time there is one line of data which may be viewed, entered or amended.

Although the system editor may allow any item in the database to be created or amended, it is not suggested that this is the general method of updating the database. Updating is usually done by BASIC programs of one form or another. The system editor is a useful tool for system administrators and programmers. It is certainly necessary to understand the basic workings of the editor if dictionaries or programs are to be created, but readers who are not going to be involved in the creation of dictionaries or programs may prefer to pass over most of the detailed information contained in this chapter.

You may have already understood that the database consists of files such as CUSTOMERS which contain records. All the data files on the system are of the same physical structure as are all the records within data files.

A record within a file consists of a key field, called an item-identifier or ITEM-ID and a number of fields or attributes. The key field must be unique within the file and the record may be accessed directly using the key field. The key field may consist of any characters, textual or numeric. The maximum number of characters in a key field is 50.

There is no restriction on the size of attributes except that the total record size is not allowed to exceed 32 Kbytes (32,767 characters).

## ACCESSING RECORDS WITH THE EDITOR

To use the editor to access any record within a file the general format of command, typed at TCL, is:

```
ED filename item-idlist
```

item-idlist will be the names of the key fields of the items required from the file. * may be used to access all of the items in the file. If some items

have been preselected from the file using SELECT or SSELECT then item-idlist may be omitted and the records SELECTed will be used. To edit the dictionary portion of a file the word DICT is used before the file name.

Suppose that we wished to edit some of the records on the customers file. Those with account numbers 310 and 392, for example. The account numbers of the customers are used as the key or item-identifying field. Once the editor is entered the following appears:

```
ED CUSTOMERS 310 392
310
NEW ITEM
TOP
.
```

Record 310 does not already exist so the word NEW ITEM are printed and the bell is rung. The cursor prompts for an editor command at the stop mark and the line pointer is set to 1. The first and subsequent attributes may be entered at this point using the insert (I) command. Each attribute can be viewed as a separate line.

Entering the command EX will exit this record without adding it to the file, i.e. record 310 still does not exist on the database, and the next record appears.

```
.EX
'310' EXITED
392
TOP
.
```

Record 392 does exist and may be viewed. The command L10 will list the first 10 lines or attributes of the record at which point line (attribute) 10 is the line which may be amended.

```
.L10
001 ACME COMPUTERS LTD
002 10 HIGH ST.]DOVER]KENT
003 DOVER 7429
004 T
005 ABC123
006 6259
007 A310]A180]A199]A200]A210]A220]A249]A256]A264]A303
008 180
```

```
009 5000]4750]4625]4075]4900]3650]5675]6925
010 2971
EOI 010
.
```

The numbers 001 to 010 to the left of this display are only used to reference the data using the editor and do not actually form part of the record. 001 is to the left of attribute 1 of the record, 002 is to the left of attribute 2 and so on.

If we had typed L22 we would have obtained the same response, since there are only ten attributes on this record, as indicated by the EOI.010. If we type L10 again, while at the bottom of the record, the editor will ring the bell as a warning that the listing is beginning again from the top.

Attributes 2, 7 and 9 have sub-fields of data on them; these sub-fields are called 'values'. These are displayed by the editor with a close square bracket (]) and actually form part of the data. They are field delimiters. If you see a backslash, /, the / is representing a sub-value delimiter. The ] and / characters are actually ASCII characters 253 and 252 respectively. They are entered from the keyboard by holding the CONTROL key down and typing ] or /.

## INSERTION

If we wish to append an attribute to the end of this record, we may do so by inserting it. Insertion begins on a new line at the end of the line currently being viewed. Simply type I for insert.

```
.I
010+ This line has been inserted
010+
.
```

In this example a new line has been inserted by typing "This line has been inserted". The line is terminated by pressing carriage return and the editor prompts with a further 010+ so that more lines may be entered. If no further lines are required, a further entry of carriage return will terminate the insertion and the editor will respond with the editor prompt character ready for the next command. The editor has indicated that the insertion is taking place after line 010 by displaying 010+ instead of 011. Line 011, should it have existed, would have become line 012 after this edit.

## THE EDITOR BUFFERS

The editor uses two buffers to create or update an item. As an editor operation is carried out on a line, the new version of the item is copied to the second buffer while updating continues on the item as it is in the first buffer. Each further amendment is carried out in the second buffer. If you wish to edit the amended item, you must first copy the contents of the second buffer back to the first buffer. The second buffer can be copied back over the first buffer by means of the F (file buffer) command. Then editing may continue on the updated version of the record.

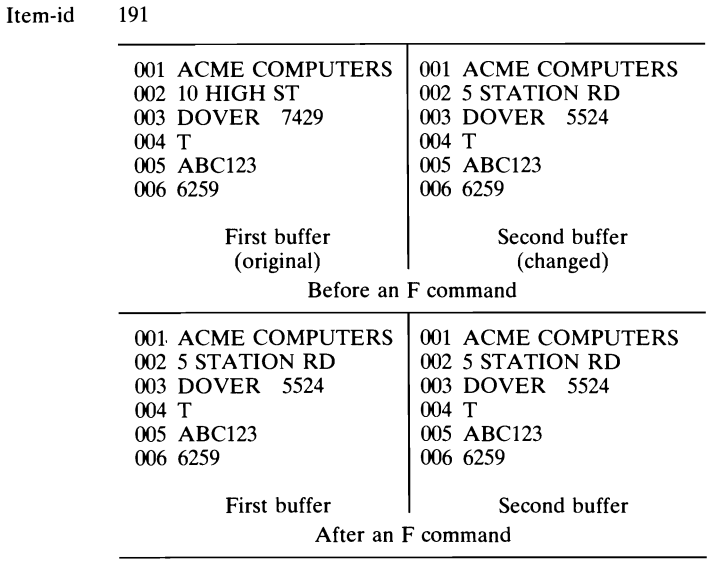



**Fig. 4.1.** The editor buffers.

As a consequence of this dual buffer system you may only edit forwards through the item. For instance, having inserted a new line after line 10 of record 392, it is not possible to make another amendment to attributes 1 to 10 inclusive before restoring the buffers with an F command.

```
.F
TOP
.
```

After executing the F (file buffer) command the editor places the record pointer back at the beginning (top) of the record and we may commence a new set of listings and changes.

**REPLACING**

Suppose that you wished to change the ABC123 to ABD234 on attribute 6. You may go straight to line 6 and replace it.

```
.G6
006 ABC123
.R
006 ABD234
.F
TOP
.G6
006 ABD234
.
```

We need not have replaced the whole line, however. We could have typed:

```
.R/C123/D234
```

and the editor would have responded:

```
006 ABD234
.
```

Note that if C123 had not been unique within the line, only the first occurrence of C123 would have been changed to D234. Also if either of the strings C123 or D234 had contained a backslash then any delimiter of our choosing may have been used instead. Further, all editor command letters may be typed in upper or lower case so r?C123?D234 means the same as R/C123/D234.

To append onto the end of an attribute, rather than inserting a new line, we use the fact that the editor considers the end of the line to be an infinite number of spaces, even though this is not really the case! So to add XYZ to the end of attribute 6 we type:

```
.g6
006 ABD234
.R/          /XYZ
006 ABD234XYZ
.F
TOP
.
```

To insert at the beginning of an attribute we use the null string, viz:

```
.G6
006 ABD234XYZ
.R//123
006 123ABD234XYZ
.F
TOP
```

and we utilise the null string to delete charactes from a line:

```
.G6
006 123ABD234XYZ
.R/123ABD//
006 234XYZ
.F
TOP
```

To enter a null attribute we type an attribute mark. This is ASCII character 254 and is the character which Pick uses to separate each attribute of data on the record. The editor does not display the attribute marks at the end of each line but it is possible to enter them by typing CONTROL up arrow (^). If we include an attribute mark in our replace command the effect is to truncate the rest of the attribute. On some systems the data is split into two attributes (IBM and Ultimate) but, on most, the rest of the data is thrown away. Note that in the following example ^ represents an attribute mark.

```
.G6
006 234XYZ
.R/3/^
006 2
.F
TOP
```

Entering an attribute mark during insertion gives empty fields:

```
.B (this command takes us to the bottom)
EOI 011
.I
011+^^^^
011+
.F
TOP
```

```
.G11
011 This line has been inserted
.L10
012
013
014
015
016
EOI 016
.
```

This is an extremely useful technique when using the editor to build dictionaries where there are usually several null attributes.

Often we wish to replace a sequence of characters which is not unique within the attribute, but nor is it the first occurrence of the sequence. To do this we can specify the range of columns on which the R command is to operate and the R command will work only on the first unique specified sequence within that range of columns.

Suppose we have an attribute:

```
001 ABCDEFGHIJDEFGNOPQ
```

and we wish to change the second occurrence of DEFG to KLM. First display the column mask by using the C command.

```
001 ABCDEFGHIJDEFGNOPQ
.C
                1         2         3         4
    12345678901234567890123456789012345678901234
.
```

From this we can see that the DEFG we want begins in column 11. We might just glance at the string and think that the DEFG we want is somewhere between columns 10 and 20. Hence we can enter:

```
.R/DEFG/KLM/10-20
001 ABCDEFGHIJKLMNOPQ
.
```

## WILD CARD CHARACTER

The up arrow character may be used as a wild card character in any editor command. This facility may be toggled by executing ^ as an editor command.

```
.^
'^' OFF
.^
'^' ON
.
```

With the up arrow on, any up arrow characters that are entered will be interpreted just as up arrows. When the up arrow is off, the wild card is on and up arrows will be interpreted as 'any character'. This is useful where sequences of characters are nearly, but not quite, the same. Suppose we had an attribute in a record which contained the data:

```
001 xax]xbx]xcx]xdx
```

If we wished to replace all of these values by null we could use four R commands and achieve this, but it would be quicker and easier if we could use the universal replace command and replace them all at once. Universal replace (RU) is the same as R except that it replaces all the occurrences instead of just the first. To replace the xs by ys in the above example we could type:

```
.G1
001 xax]xbx]xcx]xdx
.RU/x/y
001 yay]yby]ycy]ydy
.
```

To change all of these values to null, we might type:

```
.RU/x^x//
001 ]]]
```

## SEARCHING

Sometimes we wish to find or edit a specific sequence of characters within a record. This is done using the L (locate) command. Suppose we suspect that the character sequence 'xyz' occurs somewhere in the record being examined. We might type:

```
.L"xyz
```

and if xyz first occurred on line 102 the system would respond:

```
102 abcdefghijklmnopqrstuvwxyz
.
```

with the editor ready to edit that line.

If we wished to locate all the occurrences of 'xyz' in the record we could prefix the delimiter " by the number of lines to be examined.

```
.L999"xyz
```

The system would respond:

```
102 abcdefghijklmnopqrstuvwxyz
276 xyz
365 This is the third occurrence of xyz
.
```

This time the editor is pointing at line 999, because 999 lines were examined. To edit line 365 we must first 'go' to line 365.

```
.G365
365 This is the third occurrence of xyz
.
```

The delimiter character, shown as " above, may in fact be any character. A colon is a special delimiter, however. If the command above had been issued as:

```
.L:xyz
```

then the next line beginning with xyz would have been sought, the sytem responding:

```
276 xyz
.
```

## MERGING

Data may be merged from other records in the file, or from records in other files. This is achieved using the ME (merge) command.

To merge ten lines of data from line 5 of the record FRED the command is:

```
.ME10 "FRED"5
```

The system responding only with the prompt character:

```
.
```

The editor has executed the command. To view the results of the merge, the buffer must first be filed (F), and then the record listed. If

the system replies EOI 008 instead of just the prompt character, the end of the record being merged was reached at line 8, so only 4 lines were merged, 5, 6, 7 and 8.

If the "start at" field is omitted the editor will default to the beginning of the record, hence:

```
.ME10"FRED"
.
```

Will merge the first ten lines of the record FRED.

To merge data from another file we surround the file and record names with round brackets. Hence:

```
.ME10 (PERSONNEL FRED)5
.
```

will merge 10 lines from the record FRED in the PERSONNEL file, beginning at line 5 and:

```
.ME10 (DICT PERSONNEL FRED)5
```

will merge the data from the dictionary of the PERSONNEL file. If the name of the record to be merged is omitted the editor uses the name of the record being edited as the name of the record sought. This facility, in its simplest form, can be used to duplicate data already in the record being edited.

To duplicate ten lines of data from the record being edited, to the point at which the edit is taking place, the following command is used:

```
.ME10""5
```

**FILING**

We file our record using one of the file commands. We may file the record and continue editing the same record using the FS (file store) command, or we may exit the record at the same time using the FI command. Furthermore, we may change the name of the record or even file it in a different file using a syntax similar to that discussed for the merge command.

```
.FI FRED
```

would file this record under the key FRED.

```
.FI (PERSONNEL FRED
```

would file this record in the PERSONNEL file under the key FRED. Note that it is not necessary to close the brackets.

If we specified a list of key names when we started our edit, we can use the filing or exiting commands to escape from the list by appending a K (kill) command to the file instruction. Thus FIK will file the record, and kill the rest of the item-id list typed at TCL with the ED command.

The EX command is used to exit the record without filing. If the record already existed on the file, it will be left exactly as it was before the edit. Again EXK will escape any item list.

To delete the record we use the FD or FDK command. This is somewhat final and it is unfortunate that on QWERTY keyboards the D key is next to the F, making it easy to delete a record by accident.

If an FD type command is executed accidentally the record may be recovered by immediately exiting to TCL and executing the RECOVER-FD verb. This must be the next process executed because the RECOVER-FD process relies on system buffers not being overwritten. In most instances the system buffers used will be overwritten by other processes. Revelation and Prime versions of Pick are a little better in this respect because they check by asking ARE YOU SURE (Y/N)? before allowing the record to be deleted.

## PRESTORED COMMANDS

Prestored commands may be created to be equivalent to any editor command. A useful extension to the prestores is to store multiple commands by delimiting the commands using a control left square bracket (ASCII character 251) or an escape (ESC) character. This can be regarded as a 'macro'. The multiple command may then be invoked by typing in the prestore command P followed by the command number. "Editor charges" may be created by setting up a loop, during which the record being edited is filed. This is done by making the prestore command execute itself. The prestored command is then carried out on every record being edited.

Suppose we wanted to display all the occurrences of string abc in a particular file. The following sequence of instructions might be executed:

```
ED file *
XXX
top
.P1 L9999"abc[EX[P1
.P1
```

In this example P1 is set to look for the string abc, exit the record and then execute P1. When P1 is re-executed the edit is taking place on the second record and so the second record is searched. This process continues until the edit list is exhausted by there being no more records left to edit and the process exits to TCL.

Prestore command zero (P0 or simply P) is set to L22 automatically when you enter the editor. You may reset any P command at any stage. The prestore commands will remain operative until you exit the editor. That is, they remain in operation as set while you transfer from item to item. Only by exiting (e.g. to TCL) do the prestore commands get reset.

The foregoing is not an exhaustive description of all editor commands. A full description of each editor command can be found in the Pick Editor Reference Manual. A summary is presented in Appendix 1 so that reference to the correct section may be made readily.

# Chapter 5
# Building Dictionaries

The Access enquiry language is dictionary driven. That is, every word which can form part of an Access sentence will be found in a dictionary. Thus words like LIST, SORT, BY and WITH will have definitions held in an account's "master dictionary". These words have meaning when used to enquire upon any data file accessed from the account. Words such as NAME, ADDRESS and AGE, referred to in the chapter on the use of Access, have definitions held in dictionaries associated with a specific file. In the chapter on Access, the file was called PERSONNEL.

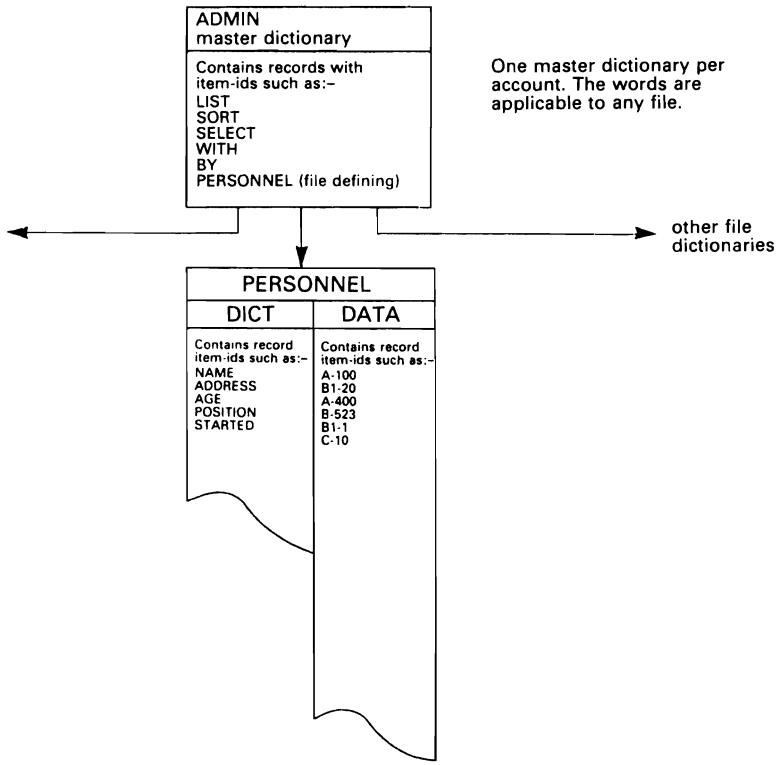



**Fig. 5.1.** Records in the dictionary define the structure of records in the data portion.

Every data file has a dictionary associated with it. The dictionary contains items which are used by Access to display data. Thus any dictionary will define the structure of the associated data file.

## THE STRUCTURE OF DICTIONARY RECORDS

Dictionary items are created using the system editor with the command:

```
ED DICT filename dictname
```

and have a fixed format:

| Attribute | Function |
|---|---|
| Item-id | Name (free format). |
| 1 | A (attribute defining) or S (synonym). |
| 2 | The number of the attribute being referenced. |
| 3 | The column heading. |
| 4 | Controlling and dependent value indicator. |
| 5 | |
| 6 | |
| 7 | Conversions to be carried out (for example dates). |
| 8 | Functions (for example arithmetic (CORRELATIVES)). |
| 9 | Justification (L (left) R (right) T (text) U (unconditional)). |
| 10 | Width of field when displayed. |

You must have a dictionary definition for each word that is subsequently employed in an Access sentence. Thus in the Access sentence:

```
LIST PERSONNEL NAME
```

the word NAME appears in the dictionary of the PERSONNEL file as a key. The dictionary definition making up the NAME record then determines exactly what data is displayed by the command.

The simplest type of dictionary definition will define a field or attribute of information of any record from the data section of the file. Thus a dictionary definition called NAME may contain the information shown in Fig. 5.2:

The first field, being A or S, is recognised by the operating system as meaning that this record may be used as a dictionary definition. There is no operational difference between A type and S type definitions.

```
ED DICT PERSONNEL NAME
NAME
TOP
.L22
001 A
002 1
003 Name of employee
004
005
006
007
008
009 L
010 20
EOI 010
.
```

**Fig. 5.2.** An ordinary dictionary definition as it appears using the editor.

The second field tells the system which field or attribute of information is to be taken from each data record being displayed. In the example above the first field of information will be displayed.

The third field is used as a column heading for Access output. This is completely free format. On output the column heading will be filled out to the maximum width of the display with dots (.). If the number of characters in the column heading exceeds the width of the field, as indicated by attribute ten of the dictionary definition, then the display width will be redefined to the width of the column heading.

For example, if the width of the AGE field in the personnel file is defined as 5, the column heading (Age) will be displayed as "Age..". However, if the width of the AGE field is defined as 2, the actual display width will be 3 because there are three letters in the word "Age".

Multiple line column headings may be specified by inserting value marks. That is:

```
003 Name of]employee
```

will appear as a column heading thus:

```
Name of.....
employee.....
```

The ninth field of a dictionary definition determines the kind of justification to be applied to the data on output. The options are L for

left, R for right and T for text. U results in an unconditionally left justified field. This determines what kind of sort will be carried out on the data in an Access sentence. Right justified data is sorted in numerical sequence. Left and text justified data are sorted in alphabetical order.

Text justified data differs from left justified data in the way that line wrapping is carried out. Left justified data will be line wrapped, should the data be longer than the specified display width, by breaking the data at the specified character. Text data will also be line wrapped, but the break of the data takes place at the last space so that words are not broken in two. Unconditionally justified data is left justified, but does not obey any width restrictions, so there is no line wrap. In the following example the data is justified in different ways, but in each case the field width is defined as 15.

| Left justified | Text justified | Unconditionally justified |
|---|---|---|
| This is justifi ed to the left | This is justified to the left | This is justified to the left |

Note that text justification does not right justify as well as left justify in the way that a word processor might.

The tenth field of a dictionary definition specifies the number of characters to be used for the output width. As indicated earlier, this may be overridden if the actual width of the column heading is wider than the number specified here. Access calculates the total width of reports from the widths of the various dictionary definitions, adding 1 for each field, so that a single space will appear between each column. This is compared with the width available on the terminal or printer. If the total width is less than the width of the terminal, the report will be displayed 'across the page'. In this case each line of the report will represent a different record and each column will be the output from a single dictionary definition. If the total width of the report would be wider than the terminal, the report will be displayed 'down the page'. Each line then represents a dictionary definition with a blank line between records.

Note that to obtain a neat "down the page" format, you must make sure that each dictionary definition has the same width specified and that the column headings are already filled out with dots. Otherwise the

*Across the page format*

| NAME | POSITION | TELEPHONE |
|---|---|---|
| FRED BLOGGS | BLACKSMITH | 123 4567 |
| JOE BROWN | JOINER | 234 5678 |

*Down the page format*

NAME ..... FRED BLOGGS
POSITION..... BLACKSMITH
TELEPHONE..... 123 4567

NAME ..... JOE BROWN
POSITION..... JOINER
TELEPHONE..... 234 5678

**Fig. 5.3.** Across the page and down the page formats of Access display.

width allowed for the column headings and fields is the same as for the columnar format, and the data is not nicely justified, nor are the column headings filled out with dots automatically.

## DEFAULT REPORTS

During the chapter on Access a meaningful report was obtained by simply typing LIST PERSONNEL. The fields to appear on such a default report are specified by defining dictionary records with the keys 1, 2, 3, 4.... etc.. The data defined by the dictionary record called 1 will be displayed as the second column (after the key), the data defined by 2 will appear as the third column and so on. Apart from this fixed way of naming these definitions, they are constructed in exactly the same way as other dictionary records. The default report will contain all dictionary definitions named in this way until there is a gap in the sequence. That is, if you have dictionary definitions called 1, 2, 3 and 5, the default report will consist of the key and the data defined by dictionary definitions 1, 2 and 3, but not 5.

Default reports are obtained in a quite different way on Revelation. Revelation dictionaries contain two records, @CRT and @LPTR. These define the default reports. The list of fields to be displayed on a default report is held on attribute 3 of these records and might read NAME ADDRESS AGE POSITION etc. The default report defined by @CRT is used when listings are being sent to the monitor and the default report defined by @LPTR is used when listings are being sent to the printer.

## CONVERSIONS AND CORRELATIVES

So far we have discussed the way in which data may be defined by a dictionary definition, in order to display that data in its raw form on an Access report. Dictionary definitions may be used to manipulate the data and to combine this data with data from other fields, within the same record or from other records. These manipulations are carried out by using "conversions" and "correlatives". The techniques presented next may be used either in the conversion field or in the correlative field, but first a word on the difference between conversions and correlatives.

Conceptually a conversion is used for some change in the format of data. For instance, a date may be stored in the file as the number of days since the 31st December 1967, but we wish to display the date in one of the everyday formats such as 24 OCT 85. This is a very common use of a Pick data conversion.

Correlatives are for defining functions or translations of data from one format, like a code, into another format, the second format being defined by a record on another file. Students of relational database theory call this a JOIN.

In actual fact the important difference between a conversion and a correlative is not to do with the type of data manipulation being carried out. The crucial difference is in how Access processes conversions and correlatives.

When the Access processor interprets a dictionary definition, correlatives are processed immediately. The resulting values are then used in any sort criteria that have been specified. Conversions are processed after the sort but just before the output. To illustrate the use of this, consider the example of a date.

Since dates are held on the database as the number of days since the 31st December 1967, it is a good idea to use this format to sort the data, that is, sort the data numerically. Just before output, but after the sort, the data is converted into its external format. It follows from this that dates held in this way should have corresponding dictionary definitions that use the conversion field to tell Access how to convert the data, and also be right justified so that the data is sorted in numerical rather than alphabetical sequence.

In this way we can obtain a chronological sort. If we took a date in its real format, such as 24 OCT 85, and tried to sort this either numerically or alphabetically, we would not succeed. For example, in both cases

1 NOV 85 would precede 24 OCT 85 [1]. This is what would happen is the date conversion were to be specified in the correlative field.

## CONVERSIONS

A typical dictionary definition of a date might look like Fig. 5.4.

| | |
|---|---|
| **DATE.OF.INVOICE** | |
| **001 A** | |
| **002 3** | Attribute where the internal format date is held. |
| **003 Invoice date** | Column heading. |
| **004** | |
| **005** | |
| **006** | |
| **007 D** | Specifying a date conversion. |
| **008** | |
| **009 R** | |
| **010 11** | |

**Fig. 5.4.** A dictionary definition with a date conversion.

The D in the conversion field will carry out a date conversion. The result will be in the format 18 FEB 1985 with an abbreviated month and a four-digit year. This format can be changed. By specifying a number up to 4 after the D, e.g. D2, the number of digits in the year is altered, so D2 would output 18 FEB 85. By specifying a delimiter after this, the format is changed from an abbreviated month to a numerical month. Therefore D2- gives 18-02-85. American formats of dates are usually achieved by means of a switch in the operating system modes. Your system supplier will be able to advise you on this.

There are some date conversions which give results other than a whole date: DD will generate only the day number of the month; DM the month number of the year; DQ the quarter number; DY the year only; and D2Y will generate a two digit year. DJ will generate the day of year, so an internal date format representing the 28th August 1985 would be displayed as 240. Another date conversion which is supported, but not always documented in the manufacturer's Pick Reference manuals is DW, which gives the day of the week, 1 being Monday, 2 Tuesday

1. Alphabetically 1 is before 2. Numerically 1 NOV 85 is before 24 OCT 85 because the most significant character (a space which is equivalent to 0) is less than 2. 3 NOV 85 would also precede 24 OCT 85 in a numerical sort but would follow 24 OCT 85 in an alphabetical sort.

and so on. DWA and DMA give the day and month in alphabetical format.

The other commonly encountered conversion is a masked decimal conversion. It is recommended that decimal numbers are held on the database with implied decimal places. 25.14 would therefore be held as 2514, the number 1 being held on another record as 100. An advantage often given by zealous salesmen for doing this is that it saves a byte of disk space for each number held! The real reason is that the Access processor only carries out calculations with integer values and truncates the real numbers at the decimal point. Much inaccuracy will result on reports with calculated data derived from real numbers held on the database. This can be overcome by using a masked decimal conversion in the conversion field.

Two synonymous conversions are supported to carry out this switching between decimal and non decimal format, MD and MR. When the McDonnell Douglas computers were the only commercially available Pick systems only MD was supported. Pick implementations support MR, which means exactly the same MD, MD being retained to maintain upwards compatibility. Pick also supports ML which acts like MR but left justifies any output masks (see below). Thus a two decimal place mask is expressed by MD2 or MR2, three decimal places by MD3 or MR3.

In much the same way that the exact format of a date may be controlled by the format of the date conversion, so the presentation of numbers may be altered by extensions to the masked decimal conversion. Some possibilities are shown in Fig. 5.5.

| Conversion | Description | Stored | Output |
|---|---|---|---|
| MR2, | Commas between 000s | 100000 | 1,000.00 |
| MR24 | 4 decimal places stored rounded to 2 d.p on output | 67890 | 6.79 |
| MRZ | Print zero as a null. | 0 | |
| MR2%8 | Pad out characters with eight zeroes and 2 implied decimal places | 67890 | 00678.90 |
| MRE | Enclose negatives in <>. | –678 | <678> |
| MR2CD | Suffix negatives with CR and positives with DB with two implied d.p. | 678 | 6.78DB |
| MR###–### | Right justified format mask | A789 | A–789 |
| ML###–### | Left justified format mask | 678A | 678–A |

**Fig. 5.5.** Some examples of column mask conversions.

On Ultimate systems the format masks must be enclosed in brackets (). Format masks are not supported by McDonnell Douglas systems. A full list of the possibilities can be found in the Pick Reference manual.

Another 'family' of conversions that might be encountered are the mask character conversions (MC). Masked character conversions carry out a variety of functions such as converting data from lower to upper case, stripping out non-alpha characters or non-numeric characters and converting between decimal and hexadecimal formats.

MT conversions deal with the presentation of time. As for dates there is an internal representation of the time, the number of seconds since midnight. One second after one o'clock in the afternoon may then be stored as 46801. An MT conversion will display this as 13:00, i.e. to the minute by the 24 hour clock with hours and minutes separated by a colon. Other versions of MT allow display in the 12 hour format followed by AM or PM (MTH), or additional display of seconds (MTS) or both (MTHS).

The final type of conversion, mentioned here for completeness, is the user exit (U) conversion. This is a U followed by a four-digit hexadecimal number giving an absolute address in the operating system for the execution of an assembler subprogram on the data. These should only be used on the explicit instruction of a software or hardware supplier. Irretrievable damage can be done by a random jump into the operating system. User access to the Pick assembly language is not supported by any manufacturer.

## CORRELATIVES

Correlatives provide a powerful method of manipulating the data in files. They are used to change the data, by carrying out a calculation or extracting parts of the data or, perhaps most importantly, using the data to translate to another part of the database. This is how the relational JOIN is achieved.

### Translations

Suppose we have a customer master file with details of the customer's address, his terms of trading, credit limit, contact name and so on. It is clearly not sensible to have to duplicate this data whenever an order is booked for the customer on the orders file. However, we still wish to be able to display the customer's details when carrying out an Access listing on the Orders file. So there has to be some way of referencing the

information held on the customer master file via dictionary definitions in the dictionary of the orders file.

In fact, one piece of information will be held on the order file about the customer. This will indicate which customer the order is for and will probably be the customer's account number. No matter what form this is held in, it must be the same as the key field for the customer master file. This is used to cross reference or 'translate' to all the other information about the customer.

Suppose the customer account number is held in field six of the orders file. The dictionary definition to display the account number is therefore quite simple and might appear like the example set out in Fig. 5.6.

```
CUSTOMER
001 A
002 6
003 Account Number
004
005
006
007
008
009 R
010 14
```

**Fig. 5.6.** A dictionary definition for a customer account number.

Now suppose the name of the customer is held in field one of the customers file. To display the name rather than the account number we start by accessing the account number and translate this into the name by using the account number to cross refer to the customers file. This is represented diagrammatically by Fig. 5.7.

A translation is defined by adding a T correlative to field 8 of the dictionary definition. What we do is place a T in field 8 followed by the target file name followed by other information which tells the computer what to do if the translation cannot be completed successfully, and which field on the customers file to use if it can. The new dictionary definition, called CUSTOMER.NAME, might look like the example shown in Fig. 5.8.

The T correlative can be expressed symbolically like this:

```
Tfilename;fail parameter;input field; output field
```

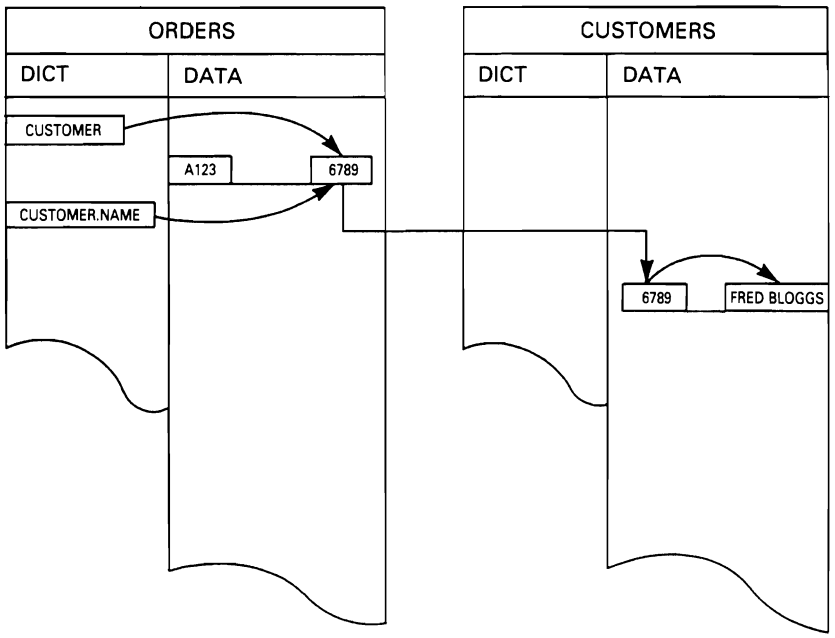



**Fig. 5.7.** A translation.

```
CUSTOMER.NAME
001 S
002 6
003 Customer name
004
005
006
007
008 TCUSTOMERS;X;;1
009 L
010 25
```

**Fig. 5.8.** A dictionary definition which defines a translation (join).

'fail parameter' tells Access what to do if the record cannot be found in the file being translated to. It can take the values B, C, V or X.

B and V failure conditions are used with Pick BASIC and their full significance is outside the scope of this book. Details can be found in the Pick Reference Manual.

| Value | Meaning |
|---|---|
| C | Use the original value instead of a translated value. |
| X | Use null instead of a translated value. |
| V | Record must exist for output, abort with error message. |
| B | Record must exist for input. |

The parameters which make up the translation are separated by semi-colons. This shows that it is not possible to translate to a file with semi-colons in the file name and so semi-colons should not be used in file names.

The third parameter in the translation tells Access which field to extract from the second file. This is called the output translation. The second parameter is not used in Access listings but may contain the same or a different field number. It is called the input translation. Translated correlatives may be used in Pick BASIC and it is when using them in this way that input translations become significant.

Suppose that we wished to display the customer's address from the orders file rather than the name. Let us suppose that the address is on field 2 of the customers file and is multi-valued. The translation does not preserve the multi-valued nature of the field. In fact the value marks, which separate the various lines of the address, are changed into spaces. Hence the address will not be displayed as multiple lines as is normal for multi-valued fields but as a single text field.

If we wished to display a single line of the address, rather than the whole of the address, we now have the problem of sorting out that line as opposed to the rest of the address. This can be done by modifying the translate correlative.

Suppose that the line to be extracted is contained in the third value. If we enter C3 rather than C, only the third value will be extracted.

| | |
|---|---|
| Field 6 of the order contains: | EB9999 |
| Field 2 of customer EB9999 is: | JOE SMITH<br>12 TOWN SQUARE<br>DULWICH |
| Correlative on the dictionary record called ADDRESS on the ORDERS file is: | TCUSTOMERS;C3;;2 |

The result of LIST ORDERS CUSTOMER ADDRESS (I) for this record is:

**EB9999 DULWICH**

### Arithmetic Correlatives

Dictionary correlatives can also be used to derive information not held anywhere on the database, calculating results from raw data held on the files being interrogated. This is done by using the A, or arithmetic, correlative.

Suppose that on the orders file there is a field containing quantities of product being ordered and another field containing the relevant unit prices of these products. The total price of each product is therefore the quantity multiplied by the unit price. There will, of course, be dictionary definitions set up describing the quantity field and the unit price field. The dictionary definition describing the total price looks like Fig. 5.9.

```
TOTAL.PRICE
001 S
002 50
003 Total]Price
004
005
006
007 MR2
008 AN(QUANTITY)*N(UNIT.PRICE)
009 R
010 8
```

**Fig. 5.9.** A dictionary definition with an arithmetic correlative.

The 50 in the second field simply describes a field which is not used on the orders file and is arbitrary. It could actually be a field which is used, but this may cause confusion. The data displayed is derived by taking the result of the dictionary name QUANTITY and multiplying this by the result of the dictionary name UNIT.PRICE. If the data in the QUANTITY and the UNIT.PRICE fields is multi-valued then each pair of multi-values will be multiplied together to give a multi-valued result. Note that if any conversion is specified in the QUANTITY and UNIT.PRICE dictionary definitions, such as MR2 suggesting that the

data is to two decimal places, then this must be reflected in a corresponding conversion in the TOTAL.PRICE definition.

```
LIST ORDERS QUANTITY UNIT.PRICE TOTAL.PRICE
```

might result in the following:

| | | | |
|---|---|---|---|
| **AA9999** | **20** | **1.50** | **30.00** |
| | **10** | **2.00** | **20.00** |
| | **12** | **3.02** | **36.24** |

Any of the arithmetic operations, +, −, *, / may be included in an A correlative as might a colon (:) meaning that the two values are to be concatenated. Dictionary definitions which themselves translate to data in other files or contain other correlatives may also be included. Constants may be specified by enclosing the constant in double or single quotes. The constant 5 is thus represented as "5". Field numbers may be included instead of dictionary names and these are specified without quotation marks. If the unit price of the products are held in attribute 22 of the orders file then

```
AN (QUANTITY)*22
```

evaluates to the same as

```
AN(QUANTITY)*N(UNIT.PRICE).
```

Sub-strings may be extracted using A correlatives with a syntax similar to the equivalent BASIC function. AN(NAME)["2","5"] returns five characters starting at character two of whatever data is described by the dictionary definition NAME.

Summations can be carried out on multi-valued data. The A correlative AS(6) will return the sum of any multi-valued data held in field 6 of the file. It is important to note that the diferent types of A correlative function may be combined so that AS(6-"10") will substract ten from each value of field six and total the result. In calculations like this, brackets may be used to indicate precedence. In the absence of any brackets the normal arithmetic precedence (multiplication and division before addition and subtraction) will be applied as in BASIC.

Details of the other A correlative functions and special operands can be found in the Pick Reference Manual. The other functions deal with the calculation of remainders and the use of logical operators. Special operands allow the introduction of system variables into calculations, such as the date or the number of items processed.

F correlatives carry out the same function as A correlatives but the

calculations must be specified in Reverse Polish notation. Other dictionary names cannot be referenced from F correlatives and these must be specified as absolute field numbers. Parameters to be pushed onto the Reverse Polish stack are separated by semi-colons. Hence F;1;2;* multiplies data from field 1 by data from field 2.

In early versions of Pick only F correlatives were supported. From the above you can see that it is not easy for people who do not have a familiarity with Reverse Polish notation to use F correlatives. A correlatives were introduced because of this, but the F correlative is still supported to maintain upwards compatibility of older applications.

The other possibilities for correlatives are used to alter or extract data, sometimes dependent on the value of a data item.

### Sub-Field Extraction

The group extraction correlative, G, extracts a part of the data surrounded by some delimiting character. The delimiting character may be any character, including a space, but not including the system delimiters, attribute mark, value mark or sub-value mark.

Suppose that field 2 of a particular record in our file contained THE QUICK BROWN FOX. We can use the G correlative to extract one or more of these words by using the spaces as delimiters. We might wish to extract the second and third words so that listing this record would return QUICK BROWN. The correlative G1 2 would achieve this. The 1 following the G says 'miss the first sub-field'. The space tells the computer that space is being used as the sub-field delimiter. The 2 following this says 'take the next two sub-fields'. This was the method used to define the SURNAME field used in the example shown in Fig. 2.6 of the chapter on Access.

### Length, Ranges and Patterns

The L correlative is used to test data to ensure that the data is of a specific length. L3 will only return data which is three characters in length. L3,5 will only return data which is between three and five characters in length inclusive. A null, or no data, will be returned for data outside these constraints.

In the same way the R correlative may be used to test data to ensure that it lies in a specific range. R3,5 will only return data with values between three and five inclusive. In addition, multiple ranges may be specified with each range separated by a semi-colon. In this way,

R1,3;5,10 will allow all values from 1 to 3 and also those values between 5 and 10. Data which does not conform to these limits will be returned as null. Note that the R correlative may only be used with numeric data.

The P correlative tests for pattern matches. The test pattern here must be enclosed in brackets so that P(3N) tests for data of exactly three numeric charcters. 3A would test for three alpha characters, or 3X for three alphanumeric characters. Specific characters may be tested for by enclosing these in quote marks and the test may be extended. Hence P('A'6N) will return any data beginning with an A and followed by six numbers.

Two rather more simple correlatives are the concatenate (C) correlative, which concatenates data together, and the text extraction correlative, T, which is used to extract a specific number of characters from data. This is not to be confused with the translate correlative which, although its code is also T, is always followed by a file name. Details can be found in the Pick Reference Manual.

### Advanced Correlatives

The S correlative tests for null or zero data and can be used to substitute other data where a null or zero is found. Two substitution parameters are required and as usual these are separated by semi-colons. The first parameter contains the data to be substituted if the existing data is non-null or non-zero. The second contains the data to be substituted if the data is null or zero. If the first parameter is * then the original data will be retained. Take the dictionary definition CREDIT.LIMIT shown in Fig. 5.10:

```
CREDIT.LIMIT
001 S
002 3
003 Credit]Limit
004
005
006
007
008 S;*;'Not set'
009 R
010 8
```

**Fig. 5.10.** A dictionary definition with a substitution correlative.

If there is a credit limit in field three of records on the file it will be shown unchanged. Otherwise the words 'Not set' will appear instead of a null value. If we had placed a 4 in place of 'Not set' then field 4 of the record would be substituted for null or zero values.

It is possible to have multiple correlatives on a single dictionary definition. A result may be pulled forwards for a subsequent operation. Each operation, a separate correlative in its own right, would be separated by a value mark in the dictionary definition. Consider the following correlative:

```
AN(DEPARTMENT):"*":N(MONTH)]TSALES.ANALYSIS;X;;1
```

In this example the sales analysis file is accessed by a key generated from the sales department name concatenated to an asterisk and the month number viz EXPORT*12. The key is generated by the A correlative. This is then used by the T correlative to access data from the SALES.ANALYSIS file. In the same way multiple T correlatives may be used to do multiple translations. Consider accessing a salesman's name, whose initals are held on the customer master file attribute 12, from the orders file on which we only hold the customer account number. The dictionary definition might be as shown in Fig. 5.11.

```
SALESMAN
001 S
002 6
003 Salesman
004
005
006
007
008 TCUSTOMERS;X;;12]TREP.DATA;X;;1
009 L
010 20
```

**Fig. 5.11.** A dictionary definition carrying out a series of translations.

S correlatives are very powerful when combined in this way with A correlatives and logical operators which can be used to test that two fields are equal or not. Consider the following correlative:

```
AN(DELIVERED)=N(ORDERED)] (S;3;4)
```

The = tests these for equality and will return a zero if the fields are unequal and a 1 if they are equal. The substitution following this results

in field 3 being displayed if the quantity delivered equals the quantity ordered, and field 4 being displayed if the two quantities are not equal.

# Chapter 6
# The Spooler

Pick has a facility for handling all output which is not to be printed on the terminal executing the job. That is, it handles output directed to a system printer or possibly the tape unit. This facility is called the spooler.

When a job is executed which directs output to a printer, such as the Access sentence LIST PERSONNEL LPTR, the process of forming the report is carried to its conclusion before any printing begins. In fact the output is directed to a temporary hold file, which cannot be edited. When this process is complete the spooler takes over and the executing line is freed to take on another job. If you observe this process the printer will appear to begin its operation at the moment that the job is complete and the executing line returns to TCL.

## THE NEED FOR A SPOOLER

There are a number of advantages to this approach. Firstly, if we consider the multi-user situation, there may be several users wishing to send output to the printer at the same time. However, there may only be one printer available on the system. All the users compete for the use of the printer. If there were no administrative system to oversee and queue the output there would be severe problems. The spooler provides this administrative system.

Secondly, because the output is being directed to a temporary hold file on disk, or in memory, the printing process may take place as quickly as any other process on the computer. Printers, especially serial printers, generally work at a much slower pace than a computer. With a spooler handling the slower process of actually printing documents and reports and the user free to continue work, a much faster and therefore better service can be given to the user.

Thirdly, as the spooler can be instructed to create a permanent hold file, rather than a temporary hold file as indicated above, and suppress the actual printing process, the printing process can be controlled by the

system administrator. The system administrator can ensure that the correct paper is loaded into the printer or reprint the report should anything unforeseen happen, such as a printer jam.

There are a whole set of commands within Pick which control the spooler. These deal with starting and stopping printers, directing output, and editing hold files.

The spooler is started automatically when the computer is first switched on and Pick is booted. If you examine the display given by the WHERE verb on a five user system with one user logged on to line zero, you might see the following display:

```
WHERE
*00 0200  FB20  121.000  121.1A2
 05 020A  BF00  170.055  170.098
 06 02C0  BF00  170.055  170.13D
```

The full significance of this display is explained in the Pick Reference Manual. We are only interested in the fact that there appear to be seven available lines on our computer, rather than five.

The sixth port, designated as a printer only port, has had a printer started and is waiting to receive output. The seventh, line 06, is the spooler, and is not in fact associated with a physical port at all. The spooler is usually designated to run on the port number one above the maximum allowable port configuration on the computer.

## STARTING AND STOPPING PRINTERS

To designate a particular line to have a system printer attached we use the STARTPTR verb. This verb is followed by a parameter list which tells the operating system how the printer is to be regarded.

The first parameter is the printer number. This is an arbitrary number between 0 and 19 and may be regarded as the 'name' by which the printer is known. Thus you can see that we can have up to 20 different printers available on the system. On computers which support them, up to 4 of these may be parallel printers.

The second parameter tells the system which jobs are to be run by the printer. In the same way that printers are named by giving them an arbitrary number, and we may have several printers on a system, we can have several jobs known as spool queues, and these are given 'names' as arbitrary numbers in the range 0 to 125. This can be envisaged like a bank. When you walk into a bank there might be several queues for the cashiers and you choose one of them. If each cashier could handle three

queues at once the analogy would be a little more vivid, for this is the capability of each designated printer line.

The third parameter tells the operating system how many pages to eject between the completion of each job. This may be any number between 0 and 9 and is very useful where reports are being spooled without producing permanent hold files since we do not usually wish our reports to run end to end without a gap.

The fourth parameter tells the operating system what type of printer is being used (serial or parallel) and where it is — that is, which line number for a serial printer or which parallel port number for parallel printers.

The fifth parameter is optional and not usually used. If present, as an A, it will initiate an alignment process sending a few lines at a time until the top of page is reached. Most modern printers have an on board procedure for setting the top of form.

The simplest STARTPTR command would be of the form:

```
STARTPTR 0, (0),1,S5
```

at which point the operating system will reply, somewhat off handedly,

**THE PRINTER CONTROL BLOCK HAS BEEN INITIALIZED**
**HOPEFULLY, THE CORRECT PAPER IS IN THE PRINTER,**
**AND THE CORRECT LPI IS SET.**

This command would be typical of a system with a single serial printer, running on line 5 with a single form feed being output at the beginning of each job.

A more complex example might be:

```
STARTPTR 3, (9,20,25),1,S24
```

Which would represent printer number 3, which is on serial port 24 and handles spool queues 9, 20 and 25. Note that the spool queue names must be enclosed in brackets so that they may be distinguished between the rest of the parameter list.

To stop a printer we refer to it by its 'name' and use the STOPPTR verb.

```
STOPPTR 3
```

would stop printer number 3. STOPPTR 0-3 would stop all the printers 0, 1, 2 and 3. STOPPTR B is global, and will stop all the printers on the system. If the printer number is suffixed by W, the printer port will return to TCL only when the printer becomes finally inactive. If

STOPPTR is executed when a print file is being output the printer will only stop when the print file is completed.

## DIRECTING OUTPUT

Output is directed at a device by means of the SP-ASSIGN verb. This must be executed before the printing process begins, so you will very often find a SP-ASSIGN executed in a Proc immediately prior to an Access statement or a report printing program. If no SP-ASSIGN has been specified, output will be directed at the system printer dealing with spool queue zero. The report will be printed as soon as the report has been formed and no hold file will be kept. The spool file will therefore disappear when the report has been printed. This default situation will be resurrected if the SP-ASSIGN verb is executed alone, with no options.

To change this the SP-ASSIGN verb is executed along with one or more options. These options direct the output to a particular device and/or spool queue, determine the timing of printing and the number of copies and report the status of the current spooler assignment. With one or two exceptions these options may be added in any combination or order.

Options which direct output to specific devices are H, S and T. The H option will create a hold file on disk, S will stop the output being directed to the printer and T will direct the output to the tape unit. A common SP-ASSIGN command is:

```
SP-ASSIGN HS
```

This will both create a hold file on disk and suppress output from being printed. If this is used the actual printing process will be totally in the control of the system administrator.

The spool queue is specified by the F option. This is immediately followed by the spool queue number. Note that no spaces are allowed between the F and the spool queue number. If you wish to direct output through spool queue 3 the correct option is F3. F 3 is wrong and will result in three copies being sent through spool queue zero. So to enqueue the printout through spool queue number three and create a hold file, the command would be:

```
SP-ASSIGN HF3
```

From this it follows that the number of copies is simply a number. It is best to surround the number with spaces so that it is clear that this is not a spool queue assignment. To instruct the last example to create two copies the command would be:

```
SP-ASSIGN HF3 2
```

The C, I and 0 options change the timing of events associated with the spooler. As indicated above, the spooler normally begins output at the end of the report formation process. If the I option is used, this is changed and the output is sent on a line-by-line basis as soon as it is printed. This does not slow the executing process down though, it simply means that the printer will begin printing before the executing process has completed the formation of the report. This only works where the output is being printed immediately. If the S option is being used to suppress the printing operation, this causes the I option to be overridden and hence have no meaning.

The C option may be used in conjunction with the I option to choke the executing process so that it does not get too far ahead of the printing process. SP-ASSIGN CI would be used in a situation where a very large report was being produced and there was a risk of running out of disk space. The C option limits the amount of disk or memory space being used to 20 frames, or 10KB. It has no meaning unless there is also an I option in effect.

The O option keeps the spool queue open at the end of the formation of the report. This is useful where a number of associated reports are to be produced one after another and keeps them together. If SP-ASSIGN HSO is executed all the reports will appear under one hold entry.

The R option is used in conjunction with BASIC programs producing reports. If a BASIC program is producing several reports simultaneously by means of the PRINT ON syntax, the R option can direct the output of each of the reports to separate spool queues. R4 would therefore affect the report being produced by PRINT ON 4, R0 would direct the output of a simple PRINT. This would normally be utilised with multiple spool queues and printers. SP-ASSIGN R4 F1 would therefore direct the PRINT ON 4 report through spool queue 1, SP-ASSIGN R0 F20 would direct the PRINT report through spool queue 20. These commands might be executed one after the other and both would be obeyed.

## MANIPULATING HOLD FILES

If the SP-ASSIGN option included H when a report was produced, a hold file will be created. To produce output from a hold file we use the SP-EDIT verb.

The simplest SP-EDIT verb will take the user through any hold files that have been produced on that account and allow the reports to be viewed and/or printed. Any reports which are currently being output will be excluded from this. To view all the hold files, from whichever account they were produced, the format SP-EDIT U is used. To view a particular hold file you may enter SP-EDIT followed by the hold file number required. SP-EDIT 4 will therefore bring up hold file number 4, if it was generated on this account.

When the SP-EDIT verb has been executed a number of prompts appear:

```
SP-EDIT
ENTRY # 1
DISPLAY (Y/N/S/D/X/(CR))?-
```

The first prompt is display. Entering Y here will display the first part of the spool file on the terminal. N will take us on to the next prompt without displaying the entry. S will take us to the spool prompt, D will take us to the delete prompt, X will exit this process and return to TCL and pressing carriage return will exit this spooler entry and go on to the next one.

```
STRING:-
```

If we enter Y or N at display the next prompt will be string. This can be used to start the report at any point. It is at its most useful where a very long report is being printed and the printer jams, say, on page 89. We do not really want to have to reprint the first 88 pages so the string prompt can be used to start the printing process off from page 88. What we do is enter a string of characters that first occur on page 88. The spooler then hunts for these characters, if it finds them it goes on to the spool prompt. If it doesn't it replies STRING NOT FOUND and goes back to the string prompt for another try. If we just press carriage return here we can go directly on to the spool prompt.

You might think that this is a very nice idea, but that the difficulty would be thinking of a character string which would not have occurred until page 88. For this reason it is highly recommended that you always

put page numbers on reports. The unique characters can then be PAGE 88!

**SPOOL (Y/N=CR/T/TN/F)?–**

An entry of Y at the spool prompt will initiate the printing process according to the current spooler assignment. N, or carriage return will pass on to the delete prompt. T will spool the report to the terminal, stopping at the end of each page. This can be aborted at the end of any page by typing control-X. TN will spool the report to the terminal but will not wait at the end of each page, returning directly to the spool prompt. F allows the report to be placed on a file in RUNOFF format.

An entry of F will result in two extra prompts:

**FILE NAME?–**

This is simply the name of the file where you wish the report to be placed. If the file does not exist, the system error message:

**[201] xxxx IS NOT A FILE NAME**

will be brought up and the whole editing process will be aborted, exiting to TCL. The second prompt is:

**INITIAL ITEM NAME?–**

A record id is required here. The report will be placed in this record in RUNOFF format. Subsequent records will be created if a page break is encountered. Suppose FRED was entered as the initial item name and the report is spooled into this item. Page 2 of the report will be filed as FRED0001, page 3 will be filed as FRED0002 and so on. The process automatically puts the RUNOFF command for chaining these items at the end of the preceding item so that the report may be recreated by the command:

```
RUNOFF file FRED
```

Note that any items which already exist with the id of FRED will be overwritten when the spooler creates the record. If there are no page break (top of form) characters in the text, such as might be produced by a BASIC program blindly printing information, a record break will be forced after 12288 characters (12 Kbytes) have been placed into a record. Any trailing spaces at the end of a page will be truncated.

If you have not asked for the hold file to be output to the printer you

are next given the opportunity to delete the hold file. If the hold file is being output the process continues with the next hold file.

**DELETE (Y/N=CR)?-**

An entry of Y will delete the hold file, N or carriage return will leave it as is.

## SP-EDIT OPTIONS

The SP-EDIT verb may be modified with many options — this is a brief summary of some of them. A full description may be found in the peripheral devices section of the Pick Reference Manual.

| Option | Meaning |
|---|---|
| n | Edit spool file n. |
| n-m | Edit files n to m. |
| Fn | Edit spool files for form queue n. |
| U | Edit all spool files. |
| accname | Edit spool files created on account accname. |
| P | Force output to the printer, despite the current SP-ASSIGN setting. |
| T | Force output to the tape, despite the current SP-ASSIGN setting. |
| R | Force output according to the current SP-ASSIGN setting, including form queue assignments. |
| MS | Spool all hold files, without prompting. |
| MD | Delete all hold files, without prompting. (Use carefully!) |

## ADMINISTERING THE SPOOLER

The above describes the way in which the spooler may be used. There are a number of verbs by means of which the system administrator may see what is happening and where, and to some extent change the activity being carried out by the spooler.

The LISTPEQS verb displays a table showing the status of all the print files on the system, as in Fig. 6.1.

The first column shows the job number as you might refer to it via the SP-EDIT verb. The second shows a rather esoteric status which is of no practical benefit. The third shows the line number which generated

the report. The CP column shows the number of copies that have been requested. The FO column shows the spool queue number that the

```
  PRINTER LIST ELEMENTS                    21 DEC 1985 12:52:30

£ STAT LK LN STATUSES          CP FO  FRMS   DATE     TIME   ACCT

1 8080     0 H C                2  2     1 20/12/84 14:12:01 SEMINAR
2 8080     0 H C                1  3     1 20/12/84 14:12:22 SEMINAR
3 8080     0 H C                1  0     2 20/12/84 14:25:09 SEMINAR
4 8080     0 H C                1  0     2 20/12/84 14:25:29 SEMINAR
5 48A5     0 P C SO L           1  0     2 21/12/84 12:51:28 SEMINAR
6 88C1     0 H  L               1  0  OPEN 21/12/84 12:52:29 SEMINAR

6 QUEUE ELEMENTS.                        8 FRAMES IN USE.
```

**Fig. 6.1.** The output produced by LISTPEQS.

report is queueing up on. The column headed FRMS shows the number of disk frames that are being utilised and is an indication of the size of the report. An entry here showing OPEN indicates that this file is being created. DATE and TIME are the date and time at which the report was produced. ACCT is the name of the account from which the reports were generated.

The STATUSES column shows exactly what is happening to this report at the moment. Here is the meaning of the codes used in the report shown in Fig. 6.1. A full list of possible codes can be found in the Pick Reference Manual.

| Code | Meaning |
|---|---|
| H | A hold file has been created. |
| P | Sent to the printer. |
| S | Spooled. |
| L | Locked (cannot SP-EDIT). |
| C | Closed (i.e. report completed). |
| O | Open (i.e. report being output). |
| (neither C nor 0) | Report not completed. |

LISTPEQS may be modified with a number of options, these are summarised as follows:

| Option | Meaning |
|---|---|
| n | display for print file n. |
| n-m | display for print files n to m. |
| acctname | display jobs created by the account acctname. |
| A | display jobs for this account only. |
| C | summary. |
| F | display print files queued for output only. |
| L | includes any deleted hold files. |
| P | output LISTPEQS to the printer. |

What LISTPEQS does for hold files, LISTPTR does for printers. The verb LISTPTR gives a table, as in Fig. 6.2, showing the status of all the available printers on the system.

```
PRINTER ASSIGNMENTS                                    13:46:43

PRINTER          OUTPUT QUEUES     PAGE  DEV OR     STATUS
TYPE   NUMBER                      SKIP  LINE £

SERIAL     0     0                    0        5    ACTIVE
SERIAL     2     1     2              1        3    INACTIVE
```

**Fig. 6.2.** The output produced by LISTPTR.

The first column here is the printer type, and may be SERIAL or PARALLEL. The second column is the printer number as allocated by the STARTPTR verb. The columns headed OUTPUT QUEUES contain the spooler queue output numbers that this printer is currently handling. The PAGE SKIP column gives the number of inter-job pages that are to be ejected at the end of each printout, again as per the STARTPTR specification. DEV will be the printer ordinal for a parallel printer or the serial line number for a serial printer. STATUS may be ACTIVE, if the printer is currently printing, INACTIVE if it is not, STOPPED if the printer has been stopped or UNALLOCATED if it has not been started, or stopped in an unusual manner. The presence of UNALLOCATED in this display could mean that there are problems to be sorted out.

Again LISTPTR may be followed by one or more of a number of options which change its operation.

SP-STATUS gives a wordier version of LISTPTR. Additionally, it indicates which job is currently being dealt with and a judgement as to whether the printer is on-line or off-line. It also indicates whether the

| Option | Meaning |
|---|---|
| n | Details of printer n only. |
| n-m | Details of printers n to m only. |
| B | Details of all possible printers whether or not they have been started. |
| P | Output LISTPTR to the printer. |

spooler is in an 'ambiguous' state. If it is, you should stop the printer causing the problem. SP-STATUS also serves another purpose. Occasionally the spooler can appear to be 'hung'; that is, jobs are queued for output but the spooler appears to be doing nothing with them. The spooler is 'asleep' and executing SP-STATUS has the effect of waking the spooler up. The same options may be used with SP-STATUS as with LISTPTR.

LISTABS may be used to show the SP-ASSIGNment of each line on the computer (see Fig. 6.3).

```
LINE   STATUS COP   FORM
  £           IES     £

  0  II         2     0
  1             0     0
  2             0     0
  3             0     0
  4             1     2
  5             0     0
```

**Fig. 6.3.** The output produced by LISTABS.

In this case the first column is the line number, the second shows the spooler assignment, the third the number of copies being generated and the fourth shows the queue number to which output would be directed.

## IF THINGS GO WRONG

Spooler problems may arise for many reasons; the problem may be as simple as having requested the wrong report to be output via SP-EDIT, or may be potentialy disastrous resulting from some hardware problem. The LISTPEQS, LISTPTR, SP-STATUS and LISTABS verbs will probably be of some help in determining the nature of the problem. SP-KILL and :STARTSPOOLER are the means of fixing the problems.

SP-KILL can abort the current printing process, or remove jobs enqueued for output and can even be used to delete printers from the system entirely. The simplest form of the SP-KILL verb will abort the current report being printed on printer zero. The report will continue for up to 512 bytes and then terminate with the message ABORT!. This is useful where a long report has been initiated by mistake and allowing this to come to a conclusion will result in a waste of both paper and time.

SP-KILL may be followed by a number of options which change its operation.

| Option | Effect |
|---|---|
| n | Kill the current output of printer n. |
| n-m | Kill the output of printers n to m. |
| B | Kill the output of all printers. |
| Fn | Stop spool file n from being output, even if it is not currently being output but is only queued for output. |
| Fn-m | Stop spool files n to m from being output. |
| FB | Stop all queued output. |

These options may be followed by one or more of the following.

| Option | Effect |
|---|---|
| A | Limit the effects to spool files created on this account. |
| N | Suppress the ABORT! message. |
| O | Stop a report currently being output. |

The F option is interesting because it gives the ability to move print files from one spool queue to another. Imagine a situation where there are two printers on the system servicing different spool queues. Using LISTPEQS you can determine that the first printer has say 20 print files to process whereas the second is inactive. Moving parts of the queue to the second printer enables you to speed up the printing process. Care has to be exercised though because KILL means exactly that. If no hold file has been created the report will be lost.

Suppose that a particular report that you wanted immediately was queuing in spool queue 6 behind another very large report which was likely to take an hour to print. When the report that you want was produced, the application reported that the printout was being generated into hold entry number 10. LISTPEQS confirms that the report is queued for output. To dequeue the report execute the command:

```
SP-KILL F10
```

**PRINT FILE # 10 WAS UNLINKED AND IS AVAILABLE AS A HOLD FILE.**

You might know that the printer which is set up to handle spool queue number 1 is free at the moment. The next thing to do is set your own SP-ASSIGN status to output to spool queue number 1 and force whatever output you produce to adhere to your SP-ASSIGN status by executing the command:

```
SP-ASSIGN F1R
```

The next thing to do is to edit the hold file and spool the report:

```
SP-EDIT 10
ENTRY # 10
DISPLAY (Y/N/S/D/X/(CR))?-S
SPOOL (Y/N=CR/T/TN/F)?-Y
```

and the report will begin printing immediately on the free printer.

There is another option for SP-KILL which must be used with caution. This is the D option and deletes a printer from the system. SP-KILL D will delete printer zero, SP-KILL D1 will delete printer 1 and so on. SP-KILL DB will delete all the printers. There may be unpleasant side effects to doing this if the printer being deleted was busy. You may find that you cannot restart the printer using STARTPTR and that the whole spooler has to be restarted. In general it is better to execute a STOPPTR first. SP-KILL D should only be used when you know that the printer(s) in question are inactive.

If all else fails and the spooler refuses to behave after treatment with SP-STATUS, STOPPTR, STARTPTR and SP-KILL the guaranteed cure all is to restart the spooler using the :STARTSPOOLER verb. This is equivalent to the process which takes place when the computer is first booted, the coldstart process. It has a number of unfortunate side effects. Under certain circumstances it can be harmless, in others the entire spool queue may be lost and all the printers deleted. It might be necessary because the spooler has dropped into the system debugger, or is trying to output to a non-existent line. Hardware faults can easily affect the spooler, particularly a disk fault resulting in the spooler logging lots (many thousands) of disk errors. In this last case it is unlikely to be affecting only the spooler and a controlled shut down and a call to the maintenance engineers may be necessary anyway.

The case where the spooler drops into the debugger can be confirmed by looking at the WHERE display.

WHERE

```
*00 0200  FB20  121.000  121.1A2
 05 02A0  F300  184.1DE  170.0CF
 06 02C0  BF00  170.055  170.13D
```

Each line of this display represents a line on the system, as was explained earlier. The first column is the line number. The asterisk means that this was the line which executed the WHERE verb. The second column is the disk frame number for the beginning of the area which the operating system uses as workspace for that line. The third column is the status of the line and the subsequent columns represent the operating system address at which the line is executing and their respective return stack addresses.

The status column is broken into two parts — the first two characters tell us the type of process being carried out; this tends to be different on different releases and different computers. If the last two characters are 40, that process is in the debugger. If the spooler's status is 40, (line 6 above), this means that the spooler itself has dropped into the debugger and you will definitely have to restart the spooler. Normally it is 00.

If this happens, execute the :STARTSPOOLER verb, normally any print files being output which are not hold files will be lost. There are options to :STARTSPOOLER with successively more side effects, these are discussed in the manual. Generally if this does not get the spooler started again successfully it is time to call the engineers.

## OTHER PICK SPOOLERS

The discussion above has centred on the standard Pick spooler. The McDonnell Douglas version, while it has its roots in the same ideas, is now quite different. The Ultimate version is slightly different and differs mainly in having different verb names. These are set out in Fig. 6.4.

On McDonnell Douglas systems the SP-ASSIGN verb is very similar to the above, although a few options differ. Option I on Pick becomes option N on McDonnell Douglas. STARTPTR, STOPPTR, SP-KILL F, SP-EDIT, LISTPEQS and SP-STATUS are carried out by a menu system instigated by the verb SP-JOBS. There is no equivalent to LISTABS. Spool queues may have real names rather than numbers, spool queue 1 could be INVOICES, spool queue 2 could be ONE-PART, spool queue 3 could be TWOPART and so on. The SP-EDIT

| Pick verb | Ultimate verb |
|---|---|
| STARTPTR | SP-STARTLPTR |
| STOPPTR | SP-STOPLPTR |
| SP-EDIT | SP-EDIT |
| SP-ASSIGN | SP-ASSIGN |
| SP-ASSIGN O | SP-OPEN |
| SP-ASSIGN C | SP-CLOSE |
| SP-EDIT | SP-EDIT |
| SP-KILL | SP-KILL |
| SP-KILL Fn | SP-DEQ |
| SP-KILL D | SP-DELETELPTR |
| LISTPEQS | SP-LISTQ |
| LISTPTR | SP-LISTLPTR |
| LISTABS | SP-LISTASSIGN |
| SP-STATUS | SP-STATUS |

**Fig. 6.4.** Ultimate spooler verbs.

part of this operates like a scaled down system editor. Instead of the prompt sequence the spooler replies:

```
TOP
.
```

and you are looking at the top of the print file. The print file cannot be amended, only viewed. The command Ln lists out the next n lines on the terminal, so the whole of the hold file may be inspected. P sends the report to the printer from the point being inspected. This can cause embarrassment if you are not positioned at the top of the report so remember to execute a T (top) first. EX exits the hold file and returns to the SP-JOBS menu.

# Chapter 7
# More about the Pick Database

In theoretical terms the Pick Database is of the Relational type, that is, the System's Database Management Subsystem can perform certain tasks and the Database files may be of such format that they closely match the academic definitions of the Relational Database Model.

These tasks are termed JOIN, PROJECT and SELECT. The file formats are called first, second and third normal form. A detailed discussion of what these consist of can be found in most texts on databases[1]. It suffices here to say that JOIN, PROJECT and SELECT can be obtained by single commands using Access; first normal form is achieved easily and second and third normal forms can be generated by using additional files. JOINs and SELECTs have already been indicated in the chapter on Access. In that chapter, Fig. 2.23 is a PROJECTion. However, Pick allows the system designer to move away from the normal forms. This might be considered bad practice but the advantages that might be gained are probably worth it.

As described in Chapter 3, the Pick database is implemented on a hierarchy of accounts, files and records. This can be represented diagrammatically as in Fig. 7.1.

## CREATING FILES

Files are created by means of the CREATE-FILE utility. The syntax is:

```
CREATE-FILE filename modulo, separation modulo, separation
```

or

```
CREATE-FILE DICT filename modulo, separation
```

or

```
CREATE-FILE DATA filename modulo, separation
```

DICT or DATA are used when it is only required to create a single level

1. Particularly *Database for the Small Computer User* by Tony Elbra, published by the National Computing Centre.

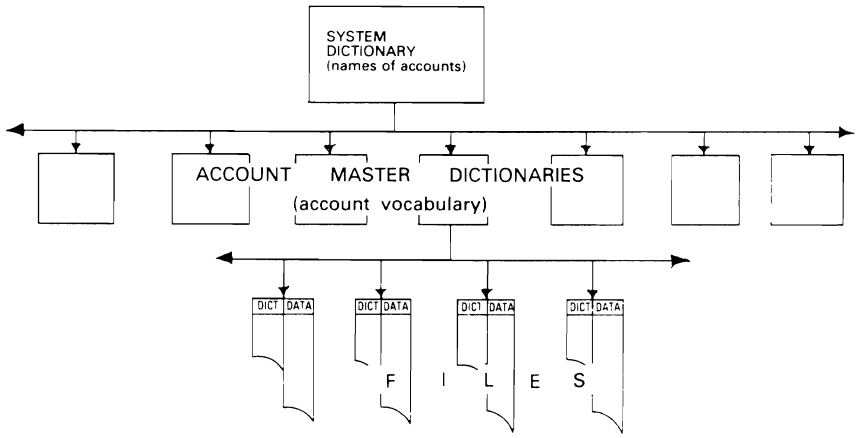



**Fig. 7.1.** The Pick File Hierarchy.

file (DICT) or to create a separate DATA portion for a already existing dictionary (DATA).

filename is the name of the file to be created. If the DICT/DATA option is omitted then a dictionary and an associated data portion are created. A file-defining item will appear in the master dictionary, which defines the dictionary portion of the file, and another will appear in the dictionary to define the data portion. These file-defining items are ordinary records but the system recognises them as pointers to file areas.

Figure 7.2 shows the structure of a file defining item, as it might appear when edited.

| | |
|---|---|
| **001 D** | .... Attribute 1 is a D |
| **002 14325** | .... File begins at disk frame 14325 |
| **003 3** | .... Modulo is 3 |
| **004 1** | .... Separation is 1 |
| **005** | .... No retrieval lock code |
| **006** | .... No update lock code |
| **007** | .... Record id conversions |
| **008** | .... Record id correlatives |
| **009 L** | .... Record id field is left justified |
| **010 10** | .... Record id field is width 10 |
| **011** | |
| **012** | |
| **013 (7,1)** | .... File will be resized to 7,1 |

**Fig. 7.2.** The structure of a file defining item.

Attributes 5 and 6 of file-defining items are used for update and retrieval locks; these are discussed at more length in the section on data security.

Attribute 13 of the a file-defining item may be used to resize a file when it is restored. The format is (modulo, separation).

Attributes 2, 3 and 4 must not be changed under any circumstances. This information is used by the operating system to calculate the position of each record in the file.

The modulo of the file tells the system how many groups of 'frames' to reserve for the file. A 'frame' is an area of disk 512 bytes long. The whole of the disk on the Pick computer is divided up into these frames, which are numbered sequentially. When the file is created the system reserves disk frames specifically for this file from its table of available contiguous frames.

The separation of the file tells the system how many contiguous frames to reserve for each group to begin with. Thus by multiplying together the modulo and the separation, this will tell you how many frames the file takes up when it is created. If the separation parameter is omitted from the CREATE-FILE command then a separation of one is assumed.

The first modulo and separation parameters in the CREATE-FILE processor define the dictionary size; the second set defines the data portion. For a single level file the second parameters are omitted.

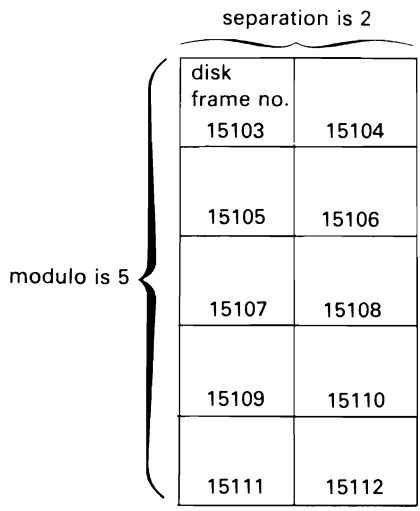



**Fig. 7.3.** Contiguous disk frames for the primary file space of a file modulo 5, separation 2.

Some attempt should be made to estimate the size of file required before it is created in terms of the number of records that are expected to be in the file and the average number of characters that are expected to be in each record. Calculate the total number of characters that are expected to be in the file, adding one for each field or subfield of information and four for each record to allow for the item length. Divide by 500 to get the number of frames required (the first twelve bytes of each frame are used by the system). Now you are in a position to estimate the best modulo and separation for the file.

The idea is to optimise the retrieval time for any particular record from the file. To do this we wish to create a situation where the data is spread evenly through the file and where overflow space is kept to a minimum, yet disk space is not used up unnecessarily. Experience seems to suggest that 125% full is about optimum. This would suggest that four out of five records will be in the primary file space, the primary file space will, theoretically, be full, and that at most only one frame fault will be required to retrieve a record. The utilisation of any file can be monitored by examining the file statistics report which is produced whenever a FILE-SAVE or ACCOUNT-SAVE is carried out.

When the system tries to retrieve a record it carries out a hashing algorithm on the record id to determine which group the record will be found in. It then begins a sequential search along the records in the group until it finds the record it wants. If it comes to the end of the frame in the primary file space it will have to frame fault — that is, jump to the overflow space indicated by this group. This is very unlikely to be contiguous to the area that is being examined currently, so a slight delay is experienced while the disk heads reposition themselves.

To ensure that this access routine behaves just as we would like it, we must ensure that the file fills up evenly. Otherwise, we could have a situation where one group is empty, and the next group is 300 or 400 per cent full. To do this we should choose a prime number for the modulo. Numbers that divide by 2, 3 and 5 provide particularly poor distributions when used as modulos.

To choose the separation, look at the expected size of any record. Obviously if a single record is more than 500 characters then there will always be a frame fault unless some action is taken to ensure that the end of the record stays in contiguous space. This is why we can set the separation. To minimise the disk space wasted, do not calculate separations as the number of frames required for one record. For instance, to allow a separation of 2 for records that are on average expected to be 600 characters in length, will result in 400 bytes of primary file space

being wasted on every group in the file. In general, it is not worth increasing the separation to 2 unless you expect records to be at least 900 characters long. Similarly, increase the separation to 3 if you expect the records to be at least 1400 characters long.

Calculation of required file size can be summarised like this:

1. Calculate the average record size expected.
2. Calculate the average number of records expected.
3. Separation is ((record size−400)/500)+1.
4. Frames required is record size * records/500.
5. Primary space required is frames * 100/125.
6. Modulo is primary space/separation.
7. Adjust the modulo to the next highest prime number.

As indicated earlier, this does not limit the size of the file in absolute terms because additional frames will be linked to the file as and when required. This procedure should optimise the performance of the system.

On a heavily-used system, overflow frames will be allocated in the usual manner from the lowest available frame in the overflow space table. The overflow space table can be displayed by the POVF utility.

```
POVF

FRAME ID                    FRAME ID
9046                   1    9100-9111          12
11327-51623        40297

TOTAL NUMBER OF FRAMES AVAILABLE 40310
```

In the example above, the first 13 frames required will come from relatively low disk space. Thereafter, frames will be taken from the area beginning at 11327. Consequently, disk head movement is greater and the system response times suffer accordingly. The situation can be improved somewhat if the system is restored. On a restore items will be taken from the tape drive and be placed in the appropriate group. When any particular group is full a frame will be taken from the lowest available space in the overflow table, but the overflow table will begin at the end of the primary file space so the disk head movement is minimised when retrieving an item which is in the overflow area. Any reallocation of file size takes place when the system is restored and this will also help. On a heavily used system a routine of regular restores (say once every three months) is desirable.

Dictionaries may have more than one data portion associated with

them. When this happens we say that the dictionary is shared between the two data files. Obviously the data files should have the same logical structure when this is required. Similarly data portions may be associated with more than one dictionary. This might be used to restrict access of parts of files to particular users who have their own privileged dictionary. When Access accesses data in files with different names than the associated dictionary it is necessary to modify the file name, separating the name of the dictionary and the data file with a comma, as in the command:

```
LIST PERSONNEL,PERSONNEL.MANAGER
```

Here the dictionary is called PERSONNEL and the data is called PERSONNEL.MANAGER. PERSONNEL.MANAGER is a data file defined in the dictionary PERSONNEL, quite separate from the PERSONNEL data file.

Similarly, when BASIC is required to OPEN one of these files the format would be:

```
OPEN '','PERSONNEL,PERSONNEL.MANAGER' TO PERSONNEL.FILE ELSE
```

This shows that files must not contain commas in the file name.

To create multiple data files of this nature a modified form of the CREATE-FILE utility is used. To create a file with two data portions and a shared dictionary a sequence of events may take place as follows:

```
CREATE-FILE PERSONNEL 1,1 23,1
CREATE-FILE DATA PERSONNEL, PERSONNEL.MANAGER 37,1
```

The first command creates a dictionary and data file called PERSONNEL. The second creates a single level data file called PERSONNEL.MANAGER that will be referenced by the same file dictionary.

To allow data files to be accessible by more than one dictionary is easier. The USING connective is employed, i.e.

```
LIST PERSONNEL USING DICT PERSONNEL.MANAGER
```

Here the data is contained in an ordinary file called PERSONNEL. The dictionary is a quite separate file, which may well have its own associated data. Both files could have been created with the simple form of the CREATE-FILE processor. In this way the personnel manager may uncover information held in the PERSONNEL file which is restricted to senior users of the file.

## Q POINTERS

The discussion so far assumes that all the data files to be created in a particular account will only be accessed from that account. This need not be the case.

As discussed above, files are defined by the system recognising D pointers in the master dictionary and in the dictionaries. Q pointers redirect the system to look for the file-defining item in another place. This can be used to tell the system that the file is really known by another name, or that the file is really in another account, or that the file is really associated with another dictionary.

```
001 Q            .... The first attribute must be Q
002 SMITH        .... The file is in the account SMITH
003 PERSONNEL    .... The file is called PERSONNEL
004
005                   Attributes 2 to 10 may be omitted
006
007
008
009 L
010 10
```

**Fig. 7.4.** The structure of a Q pointer.

If attribute 2 is omitted, then the pointer defaults to the account to which the user is logged on. If attribute 3 is omitted, then the pointer defaults to the master dictionary. Attributes 4 to 10 have the same meaning as for D pointers.

The system utility SET-FILE creates a Q pointer to any given account called QFILE; this is used as follows:

```
SET-FILE SYSPROG NEWAC
```

This indicates that the system file NEWAC will be available from the particular account in which the SET-FILE command was executed. Subsequent commands such as:

```
LIST QFILE
```

will operate on the NEWAC file until the QFILE is reset by another SET-FILE command.

## POINTER FILES

In Chapter 3, we saw that all data on the system was held in the same way. But there is one exception. This concerns files which are to hold programs. These are ordinary files, created in the ordinary way, except that the file-defining item in the master dictionary must have a DC in attribute 1, not a D. This designates the dictionary of the file to be a "pointer file" capable of holding BASIC object code. The dictionary and data sections may be used in the ordinary way, but the Pick BASIC compiler will add program pointers to the dictionary as programs are compiled. Some manufacturers provide a system utility CREATE-PFILE to put in the DC automatically on creation, some do not and the DC must be edited in after creation.

Pointer files consist of pointers to the beginnings of either BASIC object code or saved lists. The actual object code, and the saved lists are not subject to the normal 32K limit on item size because they are merely pointed to; no item length is required since no other item may reside in the same group of frames. The system file POINTER-FILE is used in this way to collect saved lists and operate upon them.

This POINTER-FILE system does give the programmer a way to get over the 32 Kbytes item size limitation on program length. What has to be done is to change the BASIC program from an ordinary program into a saved list, but in the data portion of a pointer file. This means that both dictionary and data sections must be pointer files. The following sequence of events will achieve the desired effect.

1. Create a double pointer file by ensuring that both the dictionary and the data sections of the file are defined as type DC.
2. EDIT the large program.
3. File the program with the FIL command instead if FI. This saves the program as a list, consequently the maximum size will be 64K.

Subsequent operations such as EDIT, BASIC and so on will go through the pointer, the object code will be pointed to by the dictionary section as usual.

## OTHER FILE TYPES, DX AND DY

All that remains is to give an indication of the manipulation that may be achieved by changing the file type of the file-defining items.

Placing an X after the D or DC in a file-defining item will ensure that

the file is never saved on a backup operation. This means that if the system is subsequently restored the file will no longer exist.

Placing a Y after the D or DC in a file-defining item will ensure that the data in the file is not saved on a backup operation. On a subsequent restore the file will exist, but it will be empty.

## THE PHYSICAL LAYOUT OF THE DATABASE

The physical representation of the database is quite different to the logical representation. All of the available disk is divided up into 512-byte sections called frames. The frames each have a unique number starting at 1 and incrementing sequentially from there on.

Frames may be linked together. When this happens a record of the forward link is maintained in the first linked frame and a record of the backward link is maintained in the second linked frame. In this way any number of frames may be linked together and so data structures which cannot fit into a single frame can be accommodated.

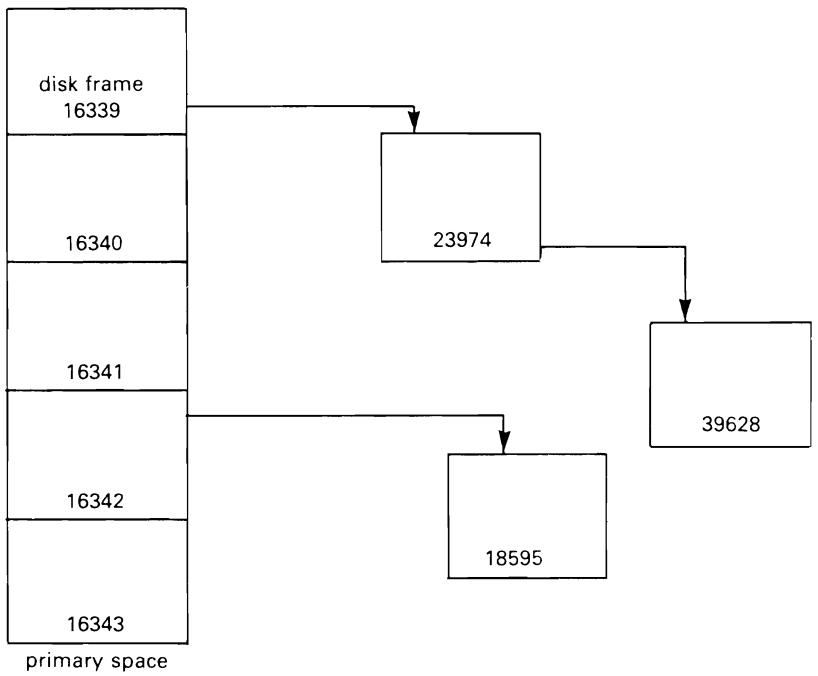



**Fig. 7.5.** A file which has one group with two overflow frames and another with one overflow frame.

The first twelve bytes of any frame contain information about the frame. Thus bytes 12–511 are available for data. More specifically:

| | |
|---|---|
| Byte 0 | — Unused. |
| Byte 1 | — Number of forward linked contiguous frames |
| Bytes 2–5 | — Next linked frame ID |
| Bytes 6–9 | — Previous linked frame ID |
| Byte 10 | — Number of backward linked contiguous frames |
| Byte 11 | — Unused |
| Bytes 12–511 | — Data |

A data record can only appear within the data portion of a frame. Physically, records are arranged with a record length, followed by the item ID followed by the record attributes. The last attribute is terminated by an attribute mark, and the end of the item is designated by a segment mark (ASCII character 255), like this:

**0029ITEM-ID^ATTRIBUTE,ONE^ATTRIBUTE,TWO^__**

If the record is the last in the group, two segment marks mark the end and the item length field will point to the second segment mark. The record length is 4 bytes long and represents the number of characters from the beginning of the record length to the end of the last attribute on the record inclusive. Thus the maximum size of a record is 32Kbytes. This restriction is to be lifted in the next major enhancement of Pick, Open Architecture.

The physical format of the data can be displayed by the DUMP verb. The syntax is

```
DUMP frameid1-frameid2 options
```

Frameid can be in decimal or, if preceded by a full stop, in hexadecimal. Options available include:

| Option | Meaning |
|---|---|
| G | The dump is to be a dump of the whole group beginning at frameid1. The display continues by dumping further frames following the forward links. |
| L | Display the links only. |
| N | Do not wait for carriage return to be pressed at the end of a page of output. |
| P | The DUMP display is to be sent to the printer. |
| X | Data is displayed with the hexadecimal ASCII codes as well as in character format. |

Figure 7.6 shows the output produced by a DUMP command with the hexadecimal option.

```
DUMP 24356 X
```

```
  FID:   24356 :   0      0      0   0  (  5F24 :  0      0      0  0 )

0001  30303539 48413132 3334FE41 6C706861      1 :0059HA1234^Alpha:
0011  20496E64 75737472 69657320 4C7464FE     17 : Industries Ltd^:
0021  32353820 48616C73 74656164 20436C6F     33 :258 Halstead Clo:
0031  7365FD52 616D7362 6F74746F 6DFD4C61     49 :se]Ramsbottom]La:
0041  6E636173 68697265 FE4D7220 536D6974     65 :ncashire^Mr Smit:
0051  682EFE35 303030FE FF303033 44485337     81 :h.^5000^_003DHS7:
0061  363534FE 5369676D 61204C74 64FE3120     97 :654^Sigma Ltd^1 :
0071  546F776E 20537175 617265FD 4C656963    113 :Town Square]Leic:
0081  65737465 72FE4D72 2042726F 776EFE31    129 :ester^Mr Brown^1:
0091  30303030 FEFF3030 35414542 39393939    145 :0000^_005AEB9999:
00A1  FE426574 6120436F 72706F72 6174696F    161 :^Beta Corporatio:
00B1  6E20496E 63FE3930 36353020 53746169    177 :n Inc^90650 Stai:
00C1  6E746F6E 20426F75 6C657661 7264FD53    193 :nton Boulevard]S:
00D1  616E2044 6965676F FD4361FD 555341FE    209 :an Diego]Ca]USA^:
00E1  42204C69 73746572 FE313030 3030FEFF    225 :B Lister^10000^_:
00F1  FFFEFFFF 31303030 30FEFFFF FE353030    241 :_^__10000^__^500:
0101  30FEFFFF FF3035FB 45454545 45454545    257 :0^___05[EEEEEEEE:
```

**Fig. 7.6.** The type of output produced by DUMP with an X option.

## GROUP FORMAT ERRORS

Any computer system can be subject to unforeseen events, such as a sudden power failure or a hardware failure. The Pick Operating System, in common with many others, does not take kindly to an uncontrolled shutdown. If such an event takes place and the computer is in the process of updating a record, the record might not be written away in exactly the correct format. Consequently the data cannot be read properly and appears to be corrupted. We call this problem a 'group format error'.

This is not to say that group format errors cannot be coped with. Indeed it is the mark of a good operating system that such serious problems are rare, but can be fixed.

A group format error (GFE) occurs when an item length appears to be incorrect, or forward/backward links are incorrect. It can also occur in process workspace, as a 'phantom' GFE. A group format error is probably the nastiest problem that can be encountered on a Pick system. Fortunately, it is probably one of the rarest.

What is of concern is that encountering a group format error can lose user data and can require a great deal of technical expertise to sort out. It almost certainly means that something is wrong with the computer hardware or that a system mode has been corrupted.

Systems where assembler language programming is going on are big trouble where GFEs are concerned. Assembler language programming should never be carried out on live and working systems. Ideally, if assembler language programming must be done the programmer should have his own system and he can live with his own problems. This is the main reason why manufacturers are so reluctant to release the assembler and will cancel maintenance contracts where assembler language programming is carried out. Only very experienced and competent personnel should carry out assembler programming and a very good reason should be required before any assembler programming is undertaken. The system provides enough utilities to be able to solve any normal data processing requirement without recourse to the assembler.

A group format error is reported by the system. Without warning:

**GROUP FORMAT ERROR nnnnn**

will appear. nnnnn will be the frame id of the frame where the problem has occurred. At the moment that the GFE is reported, the system administrator should endeavour to stop any further processing on the system. Users should not log off at this point but simply stop. Make a note of nnnnn and DUMP the frame to try to establish what file is affected. The GFE handler will have been invoked on the process which detected the GFE and will be prompting for action. Valid actions are:

D – enter the system debugger

E – end the process and exit to TCL

F – allow the GFE handler to try to fix the GFE and the process will continue. This is not recommended! The GFE handler will simply put an end of group mark at the end of the last piece of good data. Any records which are in the same group but begin after this point will effectively be lost. To recover the lost records the GFE still has to be fixed, or the records restored from the last back up. If the data has changed it will have to be reentered.

It is best to do nothing on the terminal where the GFE has occurred but go to another terminal and DUMP the frame where the GFE has been reported.

If the file can be positively identified the question is how does the GFE manifest itself?

It is likely to be of one or more of the following classes:

(1) The item length is incorrect.

(2) The item length is not a hexadecimal number.

(3) The item does not terminate with an attribute mark – segment mark sequence.

(4) The item-id hashes into the wrong group.

(5) The frame linkages are incorrect, in this case the forward link probably points to another file, or a spooler entry. The backward link of the forward link frame may not point back to the forward linking frame.

(6) No data is identifiable as records. This is probably a phantom GFE, that is, it is in the user's workspace. Logoff the offending process and the GFE should go away.

If technical help is not immediately available and the GFE is in a file where critical data appears, the following procedure might help:

First stop all users from carrying on work. They should not log off, just stop using the computer. If the frame where the GFE is has got mixed up with the free disk space table, it could get re-used by another user and the problem will be made worse.

Then T-DUMP the file using the syntax T-DUMP ONLY filename if there are default dictionary items. This will dump most of the good items to the tape with the GFE being reported in the bad group.

Next use the editor to delete the D pointer which defines the data portion of the file (ED DICT filename filename then FD). This will mean that the frames currently being used by the file will be 'black holed' and cannot be used by any process. Furthermore, if there is any conflict over linkages (forward linking into a print file for instance), the conflict is removed. Do not try to DELETE-FILE the file because the frames used will be returned to the overflow table for reuse. If some of the frames are already being used by a print file the GFE could come back again.

Now recreate the data section of the file via

CREATE-FILE (DATA filename mod, sep

where mod and sep are the original modulo and separation of the data file. This will allocate a new set of frames to the file.

The next thing that has to be done is to fill up the data file. To do this SEL-RESTORE the file from the last back up.

To bring the file up to date as much as possible T-LOAD the file

from the T-DUMP tape that was used earlier. Use the (0) overlay option so that records that have changed will be restored correctly.

Now delete any items that were deleted from the file since the last file save. The SEL-RESTORE brought these back. Additionally, if any records that were in the group with the GFE had been changed since the file save, the changes have to be done again since these will not have been present on the T-DUMP. The only data missing now is any data which has been created since the last file save and which happened to hash into the bad group. Adequate manual procedures should exist so that this can be done.

# Chapter 8
# Pick and Security

Pick provides many facilities to ensure that the computer is secure. This is because the subject of security is approached from many different angles. The system administrator will want to ensure that his system cannot be accessed by unauthorised users, and also that authorised users may only access data to which they are entitled. Technical considerations, such as record locking, are dealt with in the chapter on BASIC.

Most of this chapter deals with the SYSTEM dictionary. The SYSTEM dictionary is the highest level file in the database hierarchy (see Chapter 3). It contains records whose item-ids are the same as the account names which are available on the computer.

The SYSTEM dictionary can be accessed as an ordinary file by logging to the SYSPROG account. Thus the following procedure may be used to edit the SYSTEM entry for the ADMIN account:

```
LOGTO SYSPROG
PASSWORD:
```

then at TCL enter

```
ED SYSTEM ADMIN
```

## PASSWORDS

The first level of security for the system is password protection, which may be applied to any account, and ought to be applied to all accounts. When a user enters a valid account name at the LOGON prompt, or tries to LOGTO another account from TCL, he will be confronted with a PASSWORD prompt. Before he will be allowed to log on to the account he must type the correct password. If he gets the password wrong he will be returned to the LOGON prompt. An unauthorised attempt to LOGTO the account from another will result in the user being returned to TCL in his original account.

A utility is provided within SYSPROG for setting or changing the passwords of accounts. This utility is invoked by typing the word PASSWORD at TCL. The program prompts for the account name and

the new password and then stores a code representing the new password in the SYSTEM dictionary.

Many systems have this kind of password protection, but it can be overridden because some facility is provided by which the user may look up the password. The Pick password system works by storing a password checksum on attribute 7 of the SYSTEM dictionary[1]. This may be edited but will give no clue as to the actual password. When the user types a password at LOGON the system takes whatever he has entered as a password and calculates a checksum. This calculated checksum is compared with the checksum stored on the SYSTEM dictionary. If they match the user is logged on, if not, he is not allowed to log on.

However, this checksum may be edited out of the SYSTEM dictionary. If this happens, the password protection is removed and no password check is carried out. Password protection will also be removed if the checksum is changed to non hexadecimal data. The unscrupulous user may then LOGTO the protected account without being hindered by a password. Nevertheless, unless he replaces the checksum when he has finished, the unauthorised access will be detected because the system administrator will notice that the password protection has disappeared from the account.

This unauthorised access will only have been possible if the user were able to gain access to the SYSTEM dictionary. Clearly the SYSTEM dictionary should be protected if any data is to be secure. SYSTEM should only be accessible from SYSPROG. If SYSPROG is properly password protected, naive users will not be able to remove password protection from accounts. This is because they will need to access SYSTEM to access the password protection. To access SYSTEM they first must LOGTO SYSPROG. If SYSPROG is password protected they cannot, without first removing password protection from SYSPROG. To do that they need to access SYSTEM — back to square one!

## UPDATE AND RETRIEVAL LOCKS

If the unauthorised user has technical knowledge, he can break out of this circle by creating a Q pointer in the master dictionary of his (authorised) account. If he can get to TCL and knows how to use the editor and the format of a Q pointer to SYSTEM, then he has enough

1. McDonnell Douglas, Revelation and Information hold the password on attribute 7 as a password, not a checksum. On these systems the password may be viewed and changed with the editor.

knowledge to do this. System administrators who think that their users will not have access to this information should consider that anyone who reads this book will be armed with enough knowledge to do this. Clearly some other mechanism is required to protect SYSTEM and this is available in the form of update and retrieval locks.

Update locks prevent unauthorised processes from updating protected information. Retrieval locks prevent unauthorised processes from even reading the data that they protect. Retrieval locks are set by placing codes in attribute 5 of a file-defining item. Update locks are set by placing codes in attribute 6 of a file-defining item. The keys to the locks are distributed automatically at LOGON time. The retrieval keys that an authorised user gains are held in attribute 5 of his SYSTEM entry. The update keys are in attribute 6. He may update any file which has locks which match his update keys, plus any file which has a subset of his update keys. Similarly for retrieval. Both the locks and the keys may be multi-character and multi-valued but there is a significant difference between multi-character codes and multi-valued codes.

If a user obtains a retrieval key of ABC he may access any file which has a lock of A, AB or ABC; if, however, his key is A(value mark)B(value mark)C the files that he may access will have keys of A, B or C. Files locked by AB, without a value mark, will be forbidden. Each of the values of the retrieval keys is regarded as a separate key and so any subset of any of the keys will be allowed.

The system of update and retrieval locks and codes enables the system administrator to protect the SYSTEM file. If the SYSTEM file is retrieval protected, then only users whose privileges allow them will be able to edit, and therefore remove, password protection. The SYSPROG account should be the only account whose retrieval key matches the SYSTEM retrieval lock. Thus only an authorised LOGTO SYSPROG will allow the SYSTEM file to be viewed. Because the password protection cannot be removed from SYSPROG, unauthorised accesses are prevented and the SYSTEM file is safe. The potential user may now create Q pointers to SYSTEM from his own account but he will still not be able to access it. He will be rewarded by the system responding with

**[210] FILE 'SYSTEM' IS ACCESS PROTECTED.**

and returning him to TCL.

Any file may be similarly protected but good security starts with the SYSTEM file. Any computer which has the SYSTEM file unprotected can be broken into because it is this file which contains the keys to the

other accounts and the password checksums. If SYSTEM is protected, the only way to break into the system is via the symbolic debugger. Only an assembler programmer would be competent to do this.

## PREVENTING ACCESS TO TCL

In some situations it may not be desirable to allow users to reach TCL. The system administrator may not want the people who are entering data to have the opportunity of using Access.

For the system administrator who wishes to stop his users getting to TCL, the system debugger poses a problem. Clearly, his menus and programs may not in themselves allow the user to reach TCL, but if the user hits the break key and types END the system will very kindly deposit him at TCL. The menus and programs may have inhibited the use of the break key. This is acceptable with mature software, where the system manager knows there are no bugs and so no situations where infinite loops or even less serious program errors may occur. The problem is that even in the best written software the occasion will arise where pressing the break key is the best way out, for instance a situation where a terminal has accidentally been switched off. For this reason the break key should only be inhibited for short periods of time.

Pick does provide an answer to this problem of the system debugger. When accounts are created, a file defining item is placed in the SYSTEM dictionary. The justification field (attribute 9) is set at L. If the system manager changes this to R, the effect is to prevent escapes to TCL from the system debugger. When the user types END in the system debugger in an account with R justification the system executes the LOGON proc for that account and does not trap to TCL. In addition, the user can be prevented from typing anything but END, OFF, G or P while in the system debugger by his being assigned lower privileges. When the account is created via the CREATE-ACCOUNT facility in SYSPROG, the system prompts for the privileges to be assigned to any user logging on to the account. Basically the choices are SYS2, SYS1 and SYS0. A user obtaining SYS2 privileges cannot be prevented from using the system debugger or any other operation once at the TCL level. A user with SYS1 privileges will not be able to use the system debugger. In addition, he will not be able to use the frame DUMP utility at TCL, initiate file saves or use the assembler and mode load (MLOAD) facilities. A user with SYS0 privileges cannot do any of these either. In addition, he will be precluded from updating his master dictionary,

creating files or using the tape facilities in any way. A system manager employing these techniques may feel that his data is safe and unauthorised update or disclosure may be prevented.

User privileges may be changed by editing the SYSTEM entry for the relevant account. They will be found on attribute 8.

## UPDATING THE ACC HISTORY FILE

One more detail of the account defining item in SYSTEM that the system manager might find useful is the ability to update the accounting (ACC) file with LOGON and LOGOFF details. This would result in an audit trail of each access of the various accounts. The ACC file will hold details of: the line which accessed the account; the time and date at which the access occurred; how long the user was logged on; the number of CPU units used; and the number of printer pages that were created. This information may be listed at any time by the system manager typing LIST ACC in SYSPROG at TCL, or LIST ACC 'SMITH#1' will give the access details for the account SMITH on line 1.

This process is initiated by having a U in the justification field of the account's SYSTEM entry. If the restart option for inhibiting access to TCL from the system debugger as discussed above is also required, then the justification field may be RU.

If the update facility is being used, the system manager should note that the system will not automatically carry out any housekeeping. The ACC file will continue to grow, adding details of each access until it is manually cleared. Failure to regularly clear the ACC file will both increase the time required for LOGON and use unnecessary disk space. If the ACC file is to be cleared by means of the CLEAR-FILE verb, then it should only be done when there are no other users on the system and should be immediately followed by logging off. This is because the ACC file also holds details of the current users on the system. If the ACC file is cleared and any routine is executed which accesses information about the current users, for instance the user exit U50BB, then all the users will appear to be logged onto an account called UNKNOWN. LISTU will return [401] NO ITEMS PRESENT because this is simply an Access listing of the ACC file.

# Chapter 9
# Archiving the Database

The subject of computer security would not be complete without some discussion on the topic of recovery procedures, i.e. what do you do when the computer breaks down and the data on the system is lost?

On a typical computer which is used in a business environment, it is essential to be able to recover from a disaster. The prospect of losing the company's data is simply unthinkable. Yet it can happen. During the life of any computer running any operating system it probably will happen. All we can do is minimise the effects of a system crash by having sensible back up procedures.

Pick provides three methods of backing up the database onto magnetic media, usually a tape device. These methods are to save the whole database, a single account, or a single file.

## SAVING THE WHOLE DATABASE

On the system administrator's account, SYSPROG, there is a verb to make a logical save of the whole database. This verb is FILE-SAVE. When this command is given, every account is saved on the tape together with all the files within those accounts. Any data in overflow space on the disk is saved with the data from the primary space. On most Pick computers a file-save tape begins with a dump of a boot program and the assembler programs making up the operating system. This is referred to as the ABS (assembler and boot strap) section. The ABS section is followed by the data from the database, account by account.

This procedure can be modified to save a selection of accounts by editing a DX into attribute 1 of the account defining items in the SYSTEM dictionary. Any account which has a DX instead of a D will be omitted from the FILE-SAVE. When using this method, be sure to return the DXs to Ds when the save has been completed, otherwise the accounts will not be saved on subsequent archives.

## RESTORING THE WHOLE DATABASE

In the event of disaster, the whole database can be restored from the initial power up of the computer. The precise details vary from computer to computer but eventually the terminal connected to line 0 will display a prompt saying OPTION followed by a list of possible options, amongst which will be F. The F option carries out a reload of the operating system and the entire database contained on a tape generated using FILE-SAVE. The disk is, in effect, reinitialised and each account and file is restored in the most economical way possible. Any data in overflow will probably still be in overflow but the overflow space will be allocated next to the primary space, so you might see an improvement in response times where files have become fragmented across the disk. Any files which have been allocated resize parameters on attribute 13 of the file definition items will be resized at this point. At the end of the restoration process all the lines will be sent to the LOGON prompt and the system will be ready for use.

These two 'side effects' of restoring mean that it is a good idea to restore the database periodically even though the data has not been lost.

## SAVING A SINGLE ACCOUNT

The ACCOUNT-SAVE verb available in SYSPROG provides a means of saving the data files associated with a single account. A tape label will be placed at the beginning of the save with the account name, the system date and your own comments, which are prompted for. The command is:

```
ACCOUNT-SAVE
```

the system prompts are:

**TAPE LABEL IF REQUIRED:-**

**ACCOUNT NAME:-**

Any data files which are accessible from the account but actually belong in another account and are only pointed to by Q pointers, will not be included in the save. The Q pointer itself will be saved, as a part of the master dictionary.

**RESTORING A SINGLE ACCOUNT**

The ACCOUNT-RESTORE verb is used to restore a single account and its data files from an ACCOUNT-SAVE tape or a FILE-SAVE tape. The format of the verb is:

```
ACCOUNT-RESTORE accountname
```

the system will prompt with:

```
ACCOUNT NAME ON TAPE:-
```

The account is restored as accountname. Note that the account may be named differently on the tape.

Both the FILE-SAVE and ACCOUNT-SAVE processes generate file statistics in the file STAT-FILE in SYSPROG. These may be listed using Access and show the size of each file and how full the files are. This information can be used to periodically resize files which get heavily into overflow and so 'tune' system performance. The STAT-FILE also records the instances of any group format errors, or bad data, that have been encountered. It is a good idea to execute the Access statement

```
LIST STAT-FILE WITH GFE > "0"
```

after every FILE-SAVE. If this does not print out the message NO ITEMS PRESENT, there are problems on the database and these should be identified and fixed before any further processing is allowed.

**SAVING A SINGLE FILE**

There is a whole range of verbs for tape handling at the lowest level. T-DUMP is an Access verb used for dumping records from any file to tape. T-LOAD will load those same records from a tape onto a data file and is also an Access verb. Usually only the simplest forms of T-DUMP and T-LOAD are used:

```
T-DUMP filename
```

or

```
T-LOAD filename
```

but because these verbs are Access verbs the whole power of Access can be brought into play to dump a selection of the records, for example:

```
T-DUMP PERSONNEL WITH AGE > "65"
```

or

```
T-LOAD PERSONNEL WITH DEPARTMENT "PRODUCTION"
```

S-DUMP can be used to dump the records to tape sorted in a particular order, which is useful where the tape is to be read on to a computer utilising sequential file structures.

```
S-DUMP PERSONNEL BY NAME
```

The other tape handling verbs deal with the control of the tape.

| | |
|---|---|
| T-ATT | Attach the tape unit. |
| T-BCK | Move the tape back one tape file. |
| T-CHK | Check a tape file for parity errors. |
| T-DET | Detach the tape unit. |
| T-EOD | Move the tape to the end of the data currently held on the tape. |
| T-FWD | Move the tape forward one tape file. |
| T-RDLBL | Read a tape label. |
| T-READ | Read the tape and dump to the terminal. |
| T-REW | Rewind the tape to the beginning. |
| T-SPACE | Move the tape forwards over multiple files. |
| T-UNLOAD | Rewind the tape and detension tape arms (Not all systems) |
| T-WEOF | Write an end of file marker. |
| T-WTLBL | Write a tape label. |

## CHECKING TAPES

Using the T-CHK verb we can check a tape for parity errors, but this does not guarantee that the data on the tape can actually be read, or that the data is sensible. There is a method to check tapes which requires the computer to read every scrap of information on the tape and this involves the use of the SEL-RESTORE verb.

SEL-RESTORE is ordinarily used to restore a single record or whole file from a FILE-SAVE or ACCOUNT-SAVE tape. The format is:

```
SEL-RESTORE filename itemlist
```

with the computer prompting:

ACCOUNT NAME ON TAPE:

FILE NAME ON TAPE:

When these have been entered the tape is searched for the account and file specified. The name of every file that is encountered during this search is printed out. When the desired file is found the items from that tape file are restored into the destination file on the database and the restore halts with a message saying how many records were restored.

If a fictitious account name and file name on tape are specified, the SEL-RESTORE process will go through the whole process as described above, but of course it will not succeed in finding the specified account or file so the whole tape will be read. It is this process which gives the surety that the tape is good and that it may be reread.

SEL-RESTORE has a number of options which change its effect slightly. These enable tape files to be referenced via a sequential number or enable restores to be commenced in the middle of a tape rather than at the beginning. These are all described in the Pick Reference Manual.

## TAPE DEVICES

So far reference has been made to a 'tape' attached to the computer. Some systems may have more than one device which may be regarded as a tape. Notably, on small systems, there may be a floppy disk unit. This will only be regarded as a tape device, i.e. used for sequential storage of files rather than a device to hold part of the database. Consequently, 'rewind' in this context means position the disk head at the 'beginning' of the disk, or sequence of files, and 'forward' and 'backward' refer to moving along the file sequence.

Where there is more than one tape device available, say a floppy disk and also a four-track ¼-inch tape unit or a nine-track ½-inch tape unit, only one of these may be logically 'attached' at once. Systems capable of these configurations are supplied with verbs to attach the correct device, usually SET-FLOP, SET-CTAPE and SET-9. Any subsequent tape operation will be directed at that device until a different tape device is attached.

# Chapter 10
# Pick BASIC

## INTRODUCTION

The main high-level programming language available with Pick is a much enhanced form of BASIC called Pick BASIC[1]. In this discussion a knowledge of Dartmouth BASIC will be assumed and we will confine ourselves to the features of the language and techniques that might be applied rather than a blow-by-blow description of each of the commands and functions available. A full description of the commands and functions can, in any case, be found in the Pick BASIC manual. A summary can be found in appendices 2 and 3.

BASIC programs are written as records in data files using the system editor. The name of the record is therefore the name of the program. It is a compiled language, object code being generated which is stored somewhere on the disk. However this is not a 'true' compiler as the object code is not the same as the native machine code. The object code is more like Pascal P code and is itself interpreted at run time. A pointer to the exact place where the object code is stored is maintained in the dictionary of the program file. For this reason program files are slightly different from ordinary files. Since the dictionaries contain pointers, we call them pointer files. The system knows that a file is a pointer file by the file-defining item in the master dictionary containing a DC instead of a D.

McDonnell Douglas systems and Revelation are different in not requiring the dictionary of the file to be a pointer file. Object code is stored as a separate record in the data portion of the file with a record id which is the program name prefixed by a dollar or pound sign. So for a program called TEST, the object code will be stored in a record called $TEST.

To compile a BASIC program we use the verb BASIC or its synonym COMPILE:

1. Known as Data BASIC on McDonnell Douglas Systems, Info BASIC on Information and R/BASIC on Revelation.

```
BASIC filename programlist
```

where programlist may be a single program name or a list of programs to be compiled. If programlist is an asterisk (*) then all the programs in the file will be compiled. The compile command may be followed by options surrounded by brackets. These are described in the BASIC reference manual and cover the production of a map of the variables, a listing to the printer, suppressing of end of line markers and so on.

Successful compilation will result in the production of the object code. The program may then be executed using the RUN verb:

```
RUN filename programname
```

The object code is then interpreted by the BASIC run-time part of Pick. The compilation stage does not take the source code down to machine code level, but only to intermediate object code which is interpreted by the run-time package. The object code is, however, reentrant. This means that should more than one user be running the program at once, only one copy of the object code will be held in memory, with all of the users sharing it.

If there were compilation errors no object code would be produced. The compiler would make some attempt to indicate what was wrong and the programmer must fix this before object code can be produced. If the compilation error has been the result of a change to an existing (working) program the original object code will remain, unaltered.

Here is a rather contrived example of a sequence of error messages from the compiler:

```
[B113] LINE 1 TERMINATOR MISSING
001 PRINT THIS IS "AN ERROR
                  ^

[B110] LINE 1 'END' STATEMENT MISSING
[B100] LINE 1 COMPILATION ABORTED; NO OBJECT CODE PRODUCED
```

The terminator missing was indicated by the printing of the offending line and the up arrow. As there was an error, compilation then ceased. The compiler does not continue to find all the errors, but it does resolve any backward references, so it might report END statement missing or a missing NEXT statement and so on. Hence you should only take notice of the first error message, the rest might be spurious.

Should we wish it, programs may be subsequently catalogued:

```
CATALOG filename programlist
```

Cataloguing places a verb definition into the master dictionary of the account in which the program was catalogued so that the program may be executed via a single word command:

```
programname
```

Programs written as subroutines must be catalogued. Once catalogued any subsequent recompilations will be automatically reflected in the catalogued programs except on McDonnell Douglas systems where catalogued programs must always be recatalogued. This is because, on McDonnell Douglas systems, the process of cataloguing takes a copy of the object code and places it into a global pointer file. Subsequent compilation only updates the dollar object code version. It is of interest to note that this method of cataloguing programs results in a performance improvement on McDonnell Douglas systems because the object code can be retrieved faster.

## VARIABLE STRUCTURES

One of the main strengths of Pick BASIC is in the variable structures that are available to the programmer. There is no such thing as real, integer or string variables. All variables are treated as strings. In arithmetic operations the string values are converted to numbers before the arithmetic operation is carried out. This process is transparent to both programmer and user unless an attempt is made to carry out an arithmetic operation on non numeric data. If this happens the run-time error message:

**LINE nnn NON NUMERIC DATA USED WHERE NUMERIC REQUIRED, ZERO USED**

is output and processing continues.

Consequently there is no need to suffix string variables with a $. The statement:

```
PRINT X
```

would print the value of X whether X was a string or a number. X may also be used interchangeably as a string and a number within the same program.

Since the BASIC is compiled, rather than interpreted, the variables are given addresses at compile time and this means that we are able to

give names of any length to variables. Hence variable names will, hopefully, be meaningful.

The rules as to the exact nature of variable naming are much the same as for any other BASIC. Variable names may not begin with a number, nor may they contain an arithmetic operator (+/−*^%&). BASIC command words such as PRINT are forbidden, although BASIC function names, such as COUNT, are allowed. The reader can decide upon the advisability of this!

Variables are not, as a matter of course, assigned a default value of zero, or null, as with many BASICs. If a variable is used in a function or computation before it has been assigned a value the error message:

**LINE nnn VARIABLE HAS NOT BEEN ASSIGNED A VALUE, ZERO USED!**

is output and processing will continue.

Constants are assigned in the usual fashion. Strings may be enclosed in single or double quotes or backslashes. The availability of three string delimiters enables the other delimiters to be assigned to variables. Numeric constants do not have to be surrounded with string delimiters. All the following are valid:

```
X = "ABC"
Y = "Steven's"
Z = 'he said "hello"'
XX = /backslash enclosed/
YY = 2
ZZ = "2"
```

Real arrays with up to two dimensions are supported. These must be dimensioned before any element is referenced via the DIM statement. Once dimensioned, the number of elements may not be changed.

### Dynamic Arrays

There is a type of array structure available within Pick BASIC which is often much more useful than a real array. This type of array is called a dynamic array. The dynamic array's usefulness stems from the fact that its structure exactly reflects the structure of records on the database. It is completely floating length. It has, potentially, three dimensions. The first dimension represents attributes, the second dimension represents values and the third dimension represents sub-values.

Any BASIC variable, including elements of real arrays, is potentially a dynamic array. As an example consider the following data held as a record on the database:

```
001 Fred Smith
002 Unit 57A]Town Square Industrial Park]Sheffield]S Yorkshire
003 B12455]B12677]B12790]B12826]B13004
004 6753/125000]6802/89695]6825/66229]6867/122000/50465
005 SYK
006 MT
```

Now suppose that this data has been read into a dynamic array, called FRED.

```
FRED<2,4,1>
```

represents attribute 2, value 4, sub-value 1 of the variable FRED, or in this case 'S Yorkshire'.

Subsets of these are supported. Hence:

```
FRED<2,1>
```

represents attribute 2, value 1 of the variable FRED, 'Unit 57A'. Any sub-values within value 1 are also included.

```
FRED<2>
```

represents the whole of attribute 2 of the variable FRED including any values and sub-values. In the example this would be 'Unit 57A]Town Square Industrial Park]Sheffield]S Yorkshire'.

If FRED were a real array then:

```
FRED(2) <1,II>
```

would represent the IIth value of the first attribute of the second element of the array FRED.

We use dynamic arrays in the same way as any other variable. Thus:

```
FRED<1,II> = 'ABC'
```

would assign the value ABC to the IIth value of attribute 1 of FRED. Further:

```
NEXT.ORDER.NO = FRED<3,II>
```

would assign whatever value FRED<3,II> had to the variable

NEXT.ORDER.NO. If II had the value 3 in the example above, this would result in NEXT.ORDER.NO being assigned the value B12790.

There are also functions for manipulating dynamic arrays. The function DELETE will delete dynamic array elements. For example:

```
FRED = DELETE(FRED,2,II,0)
```

would delete the IIth value of attribute 2 of FRED, assigning the result back to FRED. If II had the value 3 this would have the effect of deleting the third value of the address in the above example. The data which was in value 4 would become value 3, value 5 would become value 4, and so on. The rest of the data would be left unchanged. An abbreviated form of DELETE is also supported. This consists of leaving off any trailing zeroes meaning 'the whole of'. Hence:

```
FRED = DELETE(FRED,2,II,0)
```

is the same as:

```
FRED = DELETE(FRED,2,II)
```

Insertions are achieved via the INSERT function:

```
FRED = INSERT(FRED,2,1,0,"ABC")
```

which will insert the value ABC at the first value of attribute 2 of FRED. All following values are pushed out by 1, thus value 1 will become value 2 and so on. Note that there is no abbreviated form of the INSERT function. If this were carried out on the data example above the address would become:

```
ABC]Unit 57A]Town Square Industrial Park]Sheffield]S Yorkshire
```

The rest of the data would be unchanged.

If the value −1 is used for any of the parameters in DELETE, INSERT or the assignment functions then the argument −1 is taken to mean the last attribute, value or sub-value. Thus:

```
FRED<1,-1> = "ABC"
```

will add a new value (ABC) to the end of attribute 1 of FRED, no matter how many values are already on the attribute. In the example this would make the first attribute multivalued, resulting in:

```
Fred Smith]ABC
```

with all the other data unchanged.

**EXTRACT and REPLACE**

The assignment functions can also be achieved by dynamic array functions. Thus:

```
FRED<1,-1> = "ABC"
```

is exactly the same as

```
FRED = REPLACE(FRED,1,-1,0,"ABC")
```

Also

```
BIT.OF.FRED = FRED<1,1>
```

is exactly the same as

```
BIT.OF.FRED = EXTRACT(FRED,1,1,0)
```

The EXTRACT and REPLACE functions existed long before the simple assignment functions. They are retained in the language so that older applications may still be used. This is a good example of Pick's commitment to transportability.

One extremely useful dynamic array function is the LOCATE function. This is used to locate specific data within a given dynamic array.

Suppose we had a dynamic array FRED, which contained the following data:

```
ABC]EFG]GHI]JKL]MNOP
```

As usual the ] character has been used to denote a value mark so attribute 1 value 2 of FRED (FRED<1,2>) is EFG. If we wanted to locate where EFG was within the dynamic array, we may use the LOCATE function. We would write:

```
LOCATE("EFG",FRED,1;FOUND) ELSE FOUND = 0
```

The first parameter is the data to be located. The second is the name of the dynamic array to be searched. The third is the attribute number to be searched.

The program will search across the values of FRED, looking for EFG. If the third parameter had been omitted the system would have searched down the attributes, matching first attribute 1, then attribute 2 and so on. If we had followed this by another comma and then another number, we could have searched along the sub-values of a particular value.

At this stage we have determined the piece of data which is to be

searched. We follow this by a semi-colon and the name of another variable. If the value EFG is found in the dynamic array, the variable FOUND will be set to the position number in which it was found. Thus, in the example above FOUND will be set to 2 when the statement has been executed. If EFG had not been found in FRED<1>, the ELSE clause would have been executed and FOUND would have been set to zero.

An ELSE clause is mandatory in a LOCATE statement but this could be NULL (i.e. do nothing). In this case the value of the 'set' variable (FOUND) would be set to one past the last position examined. In the above example, if EFG had not been present, FOUND would have been set to 6, since 5 values would have been examined.

This is useful where the data is in no particular sequence. But sometimes the data is in sequence and we wish to preserve that sequence. It would be very useful if LOCATE could respect the sequence of data and tell us where to insert data that does not already exist, and in fact it can. This is done by adding another parameter onto the LOCATE function: a 'by' parameter.

Suppose the data we were trying to locate was FGH instead of EFG. We wish to maintain the list of data in alphabetical order and we want to end up with FGH inserted into the third value:

```
LOCATE("FGH",FRED,1;FOUND;"AL") ELSE NULL
```

will set FOUND to 3. The "AL" is the sort sequence, it stands for Ascending Left justified. The LOCATE will give up as soon as it finds a value greater than the data being considered, in this case after the second value. So the variable FOUND will be set to 3. We can now use this value to INSERT into FRED:

```
FRED = INSERT(FRED,1,FOUND,0,"FGH")
```

Other sort sequences that may be used are DL (Descending Left justified) and AR and DR (Ascending and Descending Right justified) for numeric data.

### Implementation Differences

McDonnell Douglas have implemented a different syntax for insert, delete and locate dynamic array functions. While they maintain the same syntax for insert and delete as well as their new commands, INS and DEL, they do not support the Pick LOCATE syntax. Locate on a McDonnell Douglas computer has the following syntax:

LOCATE string IN var BY sort.seq SETTING found ELSE .....

## INPUT AND OUTPUT

Pick BASIC provides a number of functions and commands to facilitate the elementary tasks of input and output. Input may be taken by utilising one or other of the forms of the INPUT statement. Output is usually produced by a form of the PRINT statement.

### INPUT

INPUT, in its rawest form, is as simple as requesting that input be taken from the keyboard and stored in a BASIC variable, for example:

```
INPUT XX
```

We may modify this to input n characters by adding an input width parameter:

```
INPUT XX,2
```

This may be used to take one character from the keyboard, followed by a carriage return or two characters without a carriage return. With the simpler form of the INPUT statement, any number of characters up to 140 may be entered. After the 140th character the system will add a carriage return. This is because the size of the input buffer is limited to 140 characters. Each character typed will be echoed back to the terminal, including the carriage return. Hence we must beware of using this form of INPUT on the bottom line of the VDU since using a simple INPUT there will result in the VDU scrolling up a line. We can suppress the echoing of the carriage return by adding a colon to the end of the INPUT statement like this:

```
INPUT XX:     or     INPUT XX,2:
```

On McDonnell Douglas systems only, we can also force a carriage return to be entered and suppress the automatic carriage return on the limited length input by appending an underline character:

```
INPUT XX,2_     or     INPUT XX,2:_
```

There is a more complex form of the INPUT statement available which can cope with cursor positioning and predisplay the value of the input variable. When used with the INPUTERR and INPUTTRAP statements it can also carry out simple validation and error handling.

The format is:

```
INPUT @(30,5):XX
```

In this example the value of XX would be displayed at column 30 row 5 of the screen. The cursor would then be repositioned at column 30 row 5 and the variable may be amended. If the user just presses return, the variable will be unchanged. Full details of the use of this family of instructions can be found in the Pick BASIC Reference Manual.

**PRINT**

In common with all BASICs, the PRINT statement provides the ability to produce output.

```
PRINT XX
```

will output the current value of the variable XX followed by a carriage return.

```
PRINT XX:
```

will suppress the output of the final carriage return. When the output is to appear on a terminal, this has the effect of leaving the cursor at the end of the output.

```
PRINT XX:'   ':YY
```

will output the current value of XX, followed by three spaces and the current value of YY and a carriage return.

```
PRINT XX,YY
```

will output the values of XX and YY in a simple columnar format. Mixtures of concatenated and columnar format are also allowed, for example:

```
PRINT XX,YY:'   ':ZZ
```

Functions and calculations may also be included in the expression:

```
PRINT XX,YY+1,INT (XX):".00"
```

**Formatting**

The simple PRINT statement is not really enough to enable the comprehensive handling of screen layouts and report formatting, although with many BASICs this is all that is provided. Two facilities are provided with Pick BASIC which make the programmer's life a lot easier. These are cursor control functions and output formatting capabilities.

**The @ Function**

The @ function provides the capability to place the cursor at predetermined cursor positions. The format is:

```
@(X,Y)
```

X being the horizontal column position across the VDU and Y being the vertical row position down the VDU. The origin (0,0) is at the top left hand corner of the VDU. In practice this is used with the PRINT statement:

```
PRINT @(20,10):"The value of X is ":X
```

The @ function will operate with any terminal for which the drivers are present within the operating system. The driver being used is controlled by the TERM setting described in Chapter 13. Many different terminal drivers may be supported including VT100, ADDS and TELEVIDEO. The actual drivers provided to the end user vary from manufacturer to manufacturer so it is best to liaise with your supplier *vis-à-vis* supported terminals. Some licensees have a table driven system so that almost any terminal may be attached and will work with no coding changes.

The @ function also supports a single parameter format @(X), i.e. column X on the current row, but many terminals will not support the resulting escape sequence so it is better to stick to the row, col syntax for transportability purposes. In addition the 'true' Pick systems support a number of terminal control functions using the @ function with a negative parameter:

| | |
|---|---|
| @(−1) | Clears the screen and places the cursor at the home position (0,0). |
| @(−2) | Places the cursor at the home position. |
| @(−3) | Clears the screen from the current cursor position to the end of the screen. |
| @(−4) | Clears the screen from the current cursor position to the end of the current line. |
| @(−5) | Starts blinking on subsequently printed data. |
| @(−6) | Turns blinking off. |
| @(−7) | Initiates a protected field (data within a protected field cannot be subsequently overwritten). |
| @(−8) | Ends a protected field. |
| @(−9) | Backspaces the cursor one character position. |
| @(−10) | Moves the cursor position up one line. |

The IBM PC implementation supports many more negative parameter @ functions than this. These cover underlining and the use of colour. As the operating system is enhanced further, many more screen control functions will be incorporated into the @ function.

These functions may only be used on terminals that have equivalent escape sequences to support the particular features. In general it may be assumed that all terminals will support @(−1) and @(−2), few will support @(−7) and @(−8) and most will support the rest.

**Formatting Output**

Pick BASIC supports format masks. Format masks are used to output data over a mask and handle justification. They are handled within a PRINT statement by following the variable to be printed by a mask. For instance:

```
PRINT XX 'L#20'
```

will print the value of the variable XX left justified and fill out the value with spaces to make a field 20 wide. If XX is longer than 20 characters it will be truncated on output to 20, although the value of XX will be left unchanged. In this instance # is special character meaning to fill with spaces. Correspondingly we may right justify the data:

```
PRINT XX 'R#20'
```

This is useful where columns of numbers are to be output.

Format masks are potentially very powerful. We may change the fill character by specifying the character to be used instead of the # sign. If XX had the value 100.00 and we wrote:

```
PRINT XX 'R=10'
```

the system would output:

```
====100.00
```

We often hold a scaled value because data is not normally held with decimal places on the database. For instance, 100.00 might be held as 10000. To cater for this, a number of decimal places may be indicated by the format mask, by prefixing the format character with the number of decimal places to be used. If XX has the value 10000:

```
PRINT XX 'R2=10'
```

will result in

```
====100.00
```

We may also extend the mask by specifying alternative spacing characters. Suppose we had an American telephone number of the form, area code (3 numeric) local code (3 numeric) local number (4 numeric). We might store this as a 10-digit number but output it in a format mask breaking up the sections with a dash (–) thus:

```
PRINT "1234567890" "L#3–#3–#4"
```

and this would produce

```
123–456–7890
```

with any extra characters truncated after the zero if left justified and before the 1 if right justified.

A variety of facilities are provided by format masks: for determining the type of negative indicator, i.e. −1 could be output as −1 or (1) or 1 DR; to output commas between thousands; to utilise a descaling factor with rounding; to suppress leading zeroes; or to output a currency symbol before the value. The use of these is fully described in the Pick BASIC Programmers Reference Manual.

On Ultimate systems the format mask is treated slightly differently in that the format mask after the justification must be enclosed in parentheses, for example:

```
PRINT XX "L#3"      Pick standard
PRINT XX "L(#3)"    Ultimate
```

On Prime Information a format function, FMT, is used:

```
PRINT FMT(XX,'L#3')
```

**Directing Output**

It is, of course, not enough to be able to direct output to the user's terminal. In any application hard copy is required. Pick BASIC allows for this by offering a facility to choose the output device. PRINTER ON will direct any subsequent output via PRINT statements to the spooler. The spooler will then direct the output to whatever device is currently assigned to the spooler. PRINTER OFF reverses this.

We can actually output to several separate reports at once by mod-

ifying the PRINT statements. A normal PRINT statement can be regarded as output to report number zero. If we modified:

```
PRINT X
```

to

```
PRINT ON 2 X
```

the output would be directed to report number 2. Up to 255 separate reports can be produced simultaneously using this method.

When the spooler receives the output it waits until all the output has been received, before sending it to the system printer. Actually the spooler waits for the spool file to be 'closed'. The spool file is automatically closed when the program completes its execution, but we can artificially close the spool file and obtain the report so produced. This is necessary where reports are being produced interactively and the user does not exit from the program between report production. The command which closes the spool file is PRINTER CLOSE.

Apart from on McDonnell Douglas systems, there is a command to direct output to the VDU regardless of the current printer assignment. The syntax is exactly the same as for PRINT, the only difference being that the command is CRT, not PRINT, for example:

```
CRT "This output is always sent to your terminal"
```

The CRT statement is undocumented in most manufacturers BASIC manuals.

### Headings and Footings

The other BASIC input and output functions are designed to make life easier, particularly when producing multiple page reports. The main problem when producing multiple page reports is usually that the program has the rather tedious task of keeping track of exactly how many print lines have been output, and controlling headings and footings in the correct format. Pick BASIC can (optionally) take care of all of this. The HEADING and FOOTING statements may be used to specify headings and footings respectively. The number of lines on the report is then controlled by the setting of the TERM command, the number of lines in the HEADING and FOOTING statements and the appearance of PAGE commands, which would override the automatic page throw produced when the page is full.

There are a few problems with adopting this approach to hard copy output. Firstly the number of lines output is controlled by the number of carriage returns that have been sent to the device. This means that we cannot utilise the @( function when sending the output to a terminal. Secondly the HEADING and FOOTING will only appear on print file 0. There is no equivalent syntax for the PRINT ON commands. Hence HEADING and FOOTING only have a limited application.

Despite these drawbacks, HEADING and FOOTING are potentially quite powerful. The syntax for FOOTING is the same as for HEADING. Here is an example of the use of the HEADING statement:

```
HEADING "This is the heading for my report"
```

and this is the simplest case. Rather like the Access HEADING modifier, control options can be included within the HEADING statement:

```
HEADING "This is the heading for my report produced on 'D'"
```

The heading will be output as specified, but the D in single quotes is regarded as an option. D is replaced by the system date in the format DD MMM YY. The single quotes are thrown away and do not appear in the heading. To get a single quote into the heading you must enter two single quotation marks together. The various options available are the same as for the Access HEADING modifier discussed earlier and include printing the date, giving extra lines, printing the page number and centring text.

## PROGRAMMING STRUCTURES

Pick BASIC is a structured programming language, that is, facilities exist within the language so that it may be used in a structured way. These facilities include structures for controlling conditions and for looping. There are also a number of 'failure' constructs. For instance, the command which reads data from the database will fail if the data which it is expecting to find is not there. Therefore READ commands have a structure which allows the programmer to specify the action taken if the READ fails.

### Conditional Constructs

There are numerous syntaxes available for the IF statement. The elementary form of the IF statement is:

```
IF condition THEN command
```

where condition is some boolean expression that the computer will evaluate as 1 or 0, i.e. TRUE or FALSE. If the condition is true, the command after the THEN will be executed, if false it will not be. Here are some examples of valid conditions:

```
X = 1
X # 1
X > 1
X < 1
X >= 1
X GT 1
X GE 1
X
X > AND Y < 0
X # '' OR Y = ''
X = '' OR (X = "A" AND Y = '')
NOT(X)
X MATCHES '1A4N'
NOT(X MATCHES '"ABC"3N')
NUM(X)
ALPHA(X)
```

A more 'structured' form of the IF statement is the multi-line IF:

```
IF condition THEN
  .....
  .....
  .....     sequence of instructions
  .....
  .....
END
```

In this example the sequence of instructions is executed if the condition is true. This is easier to read and maintain than either multiple statements on one line, or a GOSUB, or a convoluted set of GOTOs.

Extending this, we may add an ELSE clause. In the elementary case:

```
IF condition THEN command ELSE command
```

and in the structured method:

```
IF condition THEN
  ...
  ...
  ...     sequence of instructions
  ...
  ...
END ELSE
  ...
  ...
  ...     sequence of instructions
  ...
  ...
END
```

In a less elegant form, we may write:

```
IF condition ELSE
```

omitting the THEN clause altogether, although I would maintain that:

```
IF NOT (condition) THEN
```

was much better.

Where there are more than two conditions that may result, nested IF statements become laborious to program and difficult to maintain. For this reason a multiple IF is supported, called CASE. The syntax is:

```
BEGIN CASE
  CASE condition
    ...
    ...     sequence of instructions
    ...
  CASE condition.2
    ...
    ...     sequence of instructions
    ...
  CASE condition.3
    ...
  etc.
END CASE
```

If CASE 1 (or even CASE 1=1 if this is clearer), which would always

evaluate as 'true', were the last condition, it would represent a 'catch all', i.e. otherwise do this. Here is an example of the use of CASE:

```
BEGIN CASE
  CASE AGE < 6
    PRINT "Infant"
  CASE AGE < 11
    PRINT "Junior"
  CASE AGE < 16
    PRINT "Senior"
  CASE 1
    PRINT "Adult"
END CASE
```

From an efficiency point of view, CASEs should be structured so that the most commonly encountered condition to be expected should come first. This is because the system tries each case in turn until it finds one that is true. The rest are ignored and execution continues after the END CASE statement.

There is a third type of conditional statement, the computed GOTO or GOSUB. This is not as acceptable from the structured programming point of view.

If a variable, II, could have any value from 1 to 10, the programmer might find it convenient to write:

```
ON II GOSUB 0100,0200,0300,0400,0500, etc.
```

If II is 1, the subroutine beginning at the statement label 0100 is executed, if 2, the subroutine at 0200, if 3, the subroutine at 0300, and so on. This is not quite as bad as its close cousin:

```
ON II GOTO 0100,0200,0300,0400,0500, etc.
```

In this case each of the program sections beginning at 0100, 0200, etc. respectively would probably terminate with another GOTO making the code rather tortuous and hence difficult to maintain.

### Failure Conditions

There are a number of statements within Pick BASIC which must have an ELSE clause to tell the system what should happen if the command fails. These are:

```
OPEN
```

READ, MATREAD AND READV
LOCATE
READNEXT
READT AND WEOF

All of these support the multi-line construct, which must be terminated by an END. In addition they may have an optional THEN clause, for example:

```
OPEN '','PERSONNEL' TO PERSONNEL.FILE THEN
  PRINT 'PERSONNEL file opened OK'
END ELSE
  PRINT 'It is impossible to open the PERSONNEL file'
  PRINT 'Process aborted'
  STOP
END
```

## LOOPING

There are two forms of loop construct supported in Pick BASIC. One of them, FOR/NEXT, is probably available in every BASIC on the market today. The syntax is:

```
FOR var = startval TO stopval
  ...
  ...     sequence of instructions
  ...
NEXT var
```

where var is a variable name which is given the starting value indicated by startval. It is incremented by one each time around the loop. The loop is exited when var exceeds the value indicated by stopval.

FOR/NEXT is used when some operation is required to be carried out a fixed number of times. The variable var will probably be used at some time during the operation as a pointer to some data or as a counter. The generalised syntax of FOR/NEXT is:

```
FOR var = startval TO stopval STEP stepval
  ...
  ...     sequence of instructions
  ...
NEXT var
```

Here stepval is the increment. Stepval may be a positive or negative

integer or a non-integer number. If stepval is negative, the loop will be executed while var exceeds or equals stopval.

Pick BASIC also supports two interesting extensions to FOR/NEXT. These give the ability to execute a set of instructions a fixed number of times but only WHILE some condition exists or UNTIL some condition arises.

The syntax is:

```
FOR var = startval TO stopval UNTIL condition
or
FOR var = startval TO stopval WHILE condition
  ...
  ...     sequence of instructions
  ...
NEXT var
```

Or generalised:

```
FOR var = startval TO stopval STEP stepval UNTIL condition
or
FOR var = startval TO stopval STEP stepval WHILE condition
  ...
  ...     sequence of instructions
  ...
NEXT var
```

The loop is exited at the point at which var exceeds stopval or at the point at which the condition becomes true (false in the case of WHILE) whichever occurs first. If the loop is exited by virtue of the condition var will be set to its last value with which the loop was executed plus one step value (usually one). This is not true if the condition causes an exit before the loop has been executed once when var will equal startval.

### DO Loops

DO loops allow an exit upon a condition at some point during the loop rather than looping a predetermined number of times. The syntax is:

```
LOOP
  ...
```

```
    ...     pre condition processing
    ...
    UNTIL condition DO
    ...
    ...     post condition processing
    ...
REPEAT
```

Like the conditional FOR/NEXT, the UNTIL clause may be replaced by a WHILE clause. DO loops are excellent devices for processing dynamic arrays. Take the example of a customer order with a number of product lines stored as a multi-valued attribute. We wish to carry out some processing on each of the lines. One approach might be to count the number of multi-values and then handle this with a FOR/NEXT loop. Alternatively the following code may be more appropriate:

```
II = 0
LOOP
  II = II + 1
  * Take each product code in turn.
  * If none remain,
  * product code will be null.
  PRODUCT.CODE = ORDER<PRODUCT.LINE,II>
  UNTIL PRODUCT.CODE = '' DO
  ...
  ...     post condition processing
  ...
REPEAT
```

The loop exits when the product code attribute has been exhausted and this is again readily understandable and easily maintainable.

## FILE I/O AND RECORD LOCKING

### Updating Files with Pick BASIC

Because Pick BASIC supports a variable structure which exactly mirrors the physical structure of the database, reading and writing data to and from files is a very simple matter.

Essentially there are four operations that may be carried out in relation to files. Files may be opened, and data may be read, written or deleted.

### Opening Files

In common with most languages files must be opened before a file transaction may be carried out. This is achieved via the OPEN statement. We may have as many files as we like open at once with Pick BASIC and there is no need to subsequently close files once they have been finished with. There is no advantage to be gained from closing files and in fact there is no facility provided to do so.

The normal syntax of the OPEN statement is as follows:

```
OPEN ' ', 'filename' TO variable.name ELSE command
```

and from here on the file is referred to by its BASIC variable name rather than its actual name. Since the first parameter is null, the system understands us to mean the data portion of the file. To open the dictionary portion we specify the first parameter as DICT. An ELSE clause is mandatory and tells the system what to do if the file is not accessible from the account in which the program is executed. On most Pick systems, opening files is a task which takes up a considerable amount of processing power so files should only be opened once where this is possible. If file references are to be made in subroutines, as well as in main line programs, it is as well to open the file in the main line program and pass the file variable to the subroutine as a parameter, or in COMMON. Then the file need not be reopened in the subroutine.

The above syntax is not the only supported syntax but it is the one which is supported universally both by 'true' Pick systems, and by McDonnell Douglas and the look-alikes. It is also the best syntax from the point of view of readability and maintainability of the program.

On 'true' Pick systems only, the following syntax may be used:

```
OPEN 'filename' TO variable.name ELSE command
```

Here filename may be a filename alone, implying the data section, or DICT filename for the dictionary. It may also be DATA filename and again the data section would be opened.

Most systems support the syntax:

```
OPEN ' ','filename' ELSE command
```

without the TO statement. Subsequent reads and writes on the file are then achieved by specifying the READ or WRITE without the file parameter. Note that only one file may be opened at once in this manner. The file being referenced is the one that was last opened in this manner. There is of course nothing to stop us opening other files with the 'normal' syntax and referencing them in the 'normal' way, as well as having the simplified method, but I would definitely recommend that for program clarity this method if not used. In the Pick reference manual it is called the null file variable.

**Reading Data from the Database**

There are three methods of reading data from the database once a file has been opened. We may read data into a dynamic array, a real array or a single attribute variable. In all cases we must specify the file being read, via the name allocated in the OPEN statement and the key to reference the actual record required. We must also specify what should happen if the record does not exist by supplying an ELSE clause.

The syntax used for reading a dynamic array is:

```
READ var FROM filevar,keyname ELSE command
```

Here var is a Pick BASIC variable name representing a dynamic array. The whole of the record keyname is read from the file represented by filevar. Attribute 1 of the record is placed into var<1>, attribute 2 into var<2>, attribute 3 into var<3>, and so on up to the end of the record. If the null file variable is being used the filevar parameter is omitted.

For a real array we write:

```
MATREAD var FROM filevar,keyname ELSE command
```

Here var is the name of a real array which has previously been dimensioned. Again the whole of the record is read but this time attribute 1 is placed into var(1), attribute 2 into var(2) etc. Any elements left over after the end of the record are set to null. If the record contains more attributes than there are elements in the array the treatment differs on different implementations. On some machines the extra attributes are placed as a dynamic array on the end of the last element, on others the program will abort into the symbolic debugger with the message:

**LINE nnn ; MATREAD NUMBER OF ELEMENTS EXCEEDS VECTOR SIZE**

If we wish to read a single attribute from a record we must specify which attribute using READV:

```
READV var FROM filevar,keyname,attno ELSE command
```

Here var becomes a single attribute variable with the contents of attribute attno from the record. If the attribute contains values or subvalues, these will be included, hence var could still be a dynamic array. READV is a little quicker than a READ simply because there is less data transfer from disk to central memory. It is useful where a single piece of data is required from a record. However one READ is quicker than two READVs so if more than one piece of data is required READ the data rather than READVing it.

### Locking

Multi-user systems usually require a system of record locking so that the integrity of the data may be ensured. Consider the example of a stock file where one record is held per product. One of the attributes on the record represents the current stock quantity which is updated by some despatch procedure and also by a production process. User 1 despatches some of the product and the program reads the stock record in order to update it. At almost the same time user 2 is producing the product and his program reads the same record in order to update it. User 1 subtracts the delivered quantity from the stock and writes the record back to the stock file. Then user 2 adds the produced quantity to the original stock figure and writes that back to the stock file. The stock file will be incorrect because the effect of the delivery was not known when user 2 read the stock record. The stock file will hold the last written figure. In this case the stock file will look as if the delivery were never made.

We stop this happening by letting user 1 lock the record when he reads it. User 2 then has to wait until user 1 has written the record back to the file before he may read it. The programmer specifies that this happens by appending a U, for Update lock, to the READ, MATREAD or READV command. The commands then become READU, MATREADU and READVU respectively. The lock is automatically reset when the corresponding record is written back to the file. If we wish to reset the locks set in the program without writing the record we may do this via the RELEASE command. In fact if we decide not to write the record back, we must execute the RELEASE command, otherwise the record will remain locked.

If it happens that something untoward happens while the lock is set, for instance the user pushes the BREAK key and exits to TCL, the lock will still be set. To enable recovery from this situation, there is a verb on the SYSPROG account, CLEAR-BASIC-LOCKS, which will clear the locks set by any port.

### Writing to Files

As you might expect, there is a corresponding WRITE statement for each of the forms of the READ statement. Thus for dynamic arrays we write:

```
WRITE var ON filevar,keyname
```

and for real arrays:

```
MATWRITE var ON filevar,keyname
```

Each of these will create a new record on the file, called keyname. If a record called keyname already exists, it will be overwritten. The single attribute version of WRITE behaves a little differently, in that it will still create a new record if one does not already exist, but if the record does exist, only the attribute indicated will be overwritten. The rest of the record will be left alone. The syntax of WRITEV is:

```
WRITEV var ON filevar,keyname,attno
```

Again, if the null file variable is being used, the filevar parameter is omitted and if attno is −1 then var is appended onto a new attribute at the end of the record.

### Deleting Records

The DELETE statement (as opposed to the DELETE function) is used to delete whole records from files. The syntax is:

```
DELETE filevar,keyname
```

## THE TECHNICAL ASPECTS OF SECURITY

Most of the programmer's worries about security centre around preventing conflicts when updating or retrieving information from the database. The solutions will centre upon the judicious use of facilities provided by the BASIC language.

**The Need for Record Locking**

Suppose we have a situation where a change is to be made to some existing information on a customer file. The computer software is highly integrated and the customer file may be being accessed from several terminals at the same time. Orders are being entered; the sales ledger is being operated; despatches are being made and so on. Each of these functions utilises the customer master file and the credit controller wants to start changing credit limits. How does the programmer ensure that changes to the customer file do not affect the other business operations? Further, how does he ensure that two users do not attempt to make different changes to a customer's credit limit?

**Group Locking**

The READU, MATREADU and READVU statements have already been discussed in the section on BASIC file updating. Briefly, these statements lock a part of a file from further retrieval by other users while the user who has read from the file manipulates the information. The lock is released when the file is updated or this user executes a RELEASE statement. Other users wishing to access records within the same group have to wait and a periodic bell is sent to their terminal to indicate that the data is being manipulated. This is a powerful system of locking but the programmer must be aware of its limitations.

Firstly, a whole group of the file is locked. In a file of modulo 100 this would result in 1% of the file being locked. Larger files have a smaller percentage locked, but smaller files have a larger percentage locked. In the worst case, a file with a modulo of 1 will have the whole of the file locked against other users.

Following on from this, it is not a good idea to READU a record and then carry out a lot of processing on the record and file the record. The operator might go to lunch in between, leaving part of the file locked! It is recommended that an ordinary READ is carried out at the beginning of the process, then the processing should be carried out. Finally the record is reread with an update lock, amended and immediately written back.

Regrettably, this still leaves a problem to be solved. If user 1 READUs record A from a file to carry out changes and then user 2 is unlucky enough to edit the same record the editor will retrieve the record! The editor only checks record locks at file time. So if the record is filed from the editor the bell will begin to sound. It will sound until

user 1 has filed his changes, then will go right ahead and file the edited changes on top of it, so losing the changes of user 1. He who files last, files loudest! The moral of this story is that the editor should never be used on live, sensitive data unless it is strictly controlled.

### The 'Deadly Embrace'

It is not good practice to lock records from several files at once. Consider user 1 locking a record in file 1 and then another record in file 2. User 2 locks records in the same groups of the same files from another program but does it by locking file 2 and then file 1. User 1 locks file 1 and then has to stop because user 2 has locked file 2. User 2 stops because user 1 has locked file 1. This is the classical 'deadly embrace'. The only way out of this is is for the system administrator to execute the SYSPROG verb CLEAR-BASIC-LOCKS and then sort out the consequences manually. Where this situation can arise it should be tackled in one of two ways. Either the system administrator imposes the discipline of always reading files in the same order, or the semaphore locking system can be employed.

This is a system where each interactive process is given a number in the range 1 to 64. At update time the processes execute the appropriate LOCK and a semaphore will be set. This can be envisaged as a door with only one key. Only one person can have the key at any one time. Anyone else who wants the key has to wait until the first person has finished with it. In a similar way, a second process wishing to update the same file set will be stopped at the LOCK. LOCK has the advantage of having an ELSE clause so the programmer can decide what to do if the lock is set, but still suffers from the disadvantage that other processes (such as the editor) will not respect the semaphore locks. The general format of the LOCK command is:

```
LOCK lock.no ELSE command
```

Remember that semaphores have to be unset as well as set and that they should not be left set for too long. To unset a semaphore lock execute the command UNLOCK lock.no.

### Basic Coding Techniques for Record Locking

True record locking can be implemented from BASIC by using an extra file for storing locks. Suppose we wished to lock a particular record on a

file. Coding like the following will achieve this quite well, while still not protecting against the editor:

```
*
* The first section ensures that only one user may set a lock
* at a time
*
LOOP
 SET = 1
  LOCK 1 ELSE
    RQM             ; * This statement will sleep for 1 sec
    SET = 0         ; * Try again someone else was setting a lock
  END
  UNTIL SET DO
REPEAT
*
* Now check that the required record is not already locked
*
SET = 0
LOOP
 READ LOCK.ITEM FROM LOCK.FILE,FILE.NAME:"*":ID THEN
*
* Record is locked because the record is present
*
    PRINT "Locked"
  END ELSE
*
* Record is not locked so lock it
*
    WRITE USER ON LOCK.FILE,NAME:"*":ID
    SET = 1
  END
  UNTIL SET DO
REPEAT
*
* Finally release the semaphore lock to allow other users to
* check and set locks
*
UNLOCK 1
```

and then to unset the lock we simply delete the lock record:

```
DELETE LOCK.FILE,FILE.NAME:"*":ID
```

## EXTERNAL SUBROUTINE FACILITIES

Pick BASIC provides three means of transferring processing from the BASIC program to the rest of the system, other than stopping the program. We may CALL another program, we may ENTER another program or we may CHAIN to another process.

### External Subroutines, CALL

CALL allows the calling of a subroutine by another program. When the called subroutine has completed its processing, control is passed back to the calling program and processing continues at the statement after the CALL. The general syntax is:

```
CALL subname(parameter1, parameter2, ....., parameterN)
```

Subname is the name of the subroutine to be called. Both the subroutine and the calling program must be catalogued in order to do this. The subroutine must begin with a SUBROUTINE statement which matches the CALL:

```
SUBROUTINE subname(parameter1, parameter2, ....., parameterN)
```

although the variable names given to the parameters in the subroutine need not be the same as those passed. The subroutine is terminated and control passed back to the calling program by a RETURN statement.

Parameters may also be passed from program to subroutine and vice versa by a COMMON statement:

```
COMMON parameter1, parameter2, parameter3, ..... , parameterN
```

There is no limit to the length of either the COMMON statement or the passed parameter list although it is generally better to keep parameter lists short and pass data in COMMON because the COMMON statement utilises a common data area whereas a copy of the parameters is taken for the subroutine.

Programs with COMMON statements may call subroutines with no COMMON statement although the subroutine will not then be able to access any of the data in the COMMON area. The reverse is not true however, programs with no COMMON may not call subroutines with a COMMON statement as the variable map would then be corrupted. In general it is permissible to call subroutines whose COMMON statement is equal to or shorter than the calling program, but not those whose COMMON is longer than the calling program.

An interesting facility with CALL is the ability to call indirectly.

```
CALL @subname(parameters)
```

would call a subroutine whose name was held in the BASIC variable subname. This is useful where the subroutine to be called depended upon some input value, for instance. Rather than have a multiple CASE and several CALLs, the problem can be handled in a single statement. The assumption being made is that the same parameter list will be used for all the possible called subroutines.

### Transferring Control, ENTER

The ENTER statement allows control to be passed to another program. No parameter list may be passed and any data which is required to be communicated to the entered program must be placed in COMMON. Control is not passed back to the calling program when the entered program has completed processing. For this reason you must not ENTER a program from a subroutine or else the return stack maintained by the operating system will be corrupted. As for called subroutines the entered program must be catalogued and the indirect form is supported. The syntax is:

```
ENTER progname
or

ENTER @progname
```

### Transferring to Other Processes, CHAIN

CHAIN allows escape from the BASIC program to any valid TCL command. This could be an Access listing, a Proc, a system command such as WHO or another program either executed or via the RUN verb. Control is not passed back to the calling program after use. The command must be passed as a string, either directly, enclosed in quotes, or indirectly by assembling the command string beforehand into a BASIC variable and passing that.

```
COMMAND = "LISTU"
CHAIN COMMAND

or
CHAIN "LISTU"
```

If you are to chain to a BASIC program you have the option of reinitialising the variable map or, by adding an I option, to pass all the data on a one for one mapping. This is difficult to use because the first named variable in the calling program becomes the first named variable in the chained program, the second the second and so on.

Input may be stacked for any process using the DATA statement. There may be any number of DATA statements placed before the CHAIN statement which will pass data via INPUT statements in a BASIC program or TCL prompts or Proc prompts on a first in first out basis.

In this example the program chains to a copy process where the stacked input in the DATA statement contains the destination of the copy. (The COPY command is described in Chapter 13.)

```
DATA "(YESTERDAYS-INVOICES)"
CHAIN "COPY TODAYS-INVOICES *"
```

## PROGRAMMING FOR EFFICIENCY

The following are hints and explanations that will help to optimise the performance of a given application:

1. *Avoid the use of large dynamic arrays.*

Dynamic arrays are very convenient and their use is to be applauded because of their contribution to easy maintainability. However, to access the 100th attribute of a dynamic array, the system starts at the beginning and examines each character, counting the attribute marks as it goes. Having passed the 99th attribute mark, the extraction of the 100th can begin. It can be seen that this is an inefficient process.

2. *Equate real array elements.*

A similar process to retrieving dynamic array elements is applied to real arrays, but this can be circumnavigated by EQUATEing the array elements to other variable names, e.g.:

```
EQUATE TRADE.TERMS TO ARRAY(10)
```

In this case, the address of the array element, rather than the beginning of the array, will be recorded as being the address of the variable. It also has the laudable effect of making the program more readable if meaningful variable names are used.

3. *Use COMMON variables, rather than parameter lists when passing data to and from subroutines.*

COMMON lists are more efficient than parameter lists because they are not copied into a new area of workspace when the subroutine is called. Files, in particular, should be opened to COMMON variables so that they do not have to be re-opened in subroutines. The opening of files is a relatively time-consuming process. Remember that although it is permissible to call subroutines with no or smaller COMMON blocks than the calling process, it is not permissible to call a subroutine with a bigger COMMON block. This will corrupt the local variable area on return from the subroutine, causing unpredictable results!

# Chapter 11
# The PROC Job Control Language

This chapter shows how the various facilities provided by the Pick Operating System can be linked together using the job control language, PROC.

Procs provide a method of storing one or more commands which may have been entered at TCL so that those commands may be invoked with a single word command. Proc has a number of features that take it beyond the realms of a simple job controller in that there are facilities for screen formatting, taking in input and testing the validity of that input, and branching and subroutine facilities.

Procs are interpreted at run time. No compilation phase is necessary. Like BASIC programs, Procs exist as records on files and so the method of entry is via the system editor. The first attribute of any Proc must be PQ. Pick uses this to identify the record as an executable Proc. If the Proc is placed into the master dictionary of any account the Proc can then be executed by typing in the name of the Proc at TCL. The system utility LISTU is an example of a Proc which executes an Access listing.

Although Procs can pass control to any system utility, Access command or BASIC program, control will always pass back to the calling Proc when the process is terminated. Only when the Proc is exhausted, or specifically exited, is control passed back to TCL.

Procs operate by manipulating an input buffer and an output buffer. Both input and output buffers are further divided into primary and secondary buffers. There are therefore four buffers that will be operated upon.

Data that is typed in at TCL passes into the primary input buffer. Data which is used to process a particular activity is placed into the primary output buffer. A simple example will clarify this. Suppose we wished to execute the Access statement

```
SORT PERSONNEL BY NAME WITH AGE > "25" NAME AGE DEPARTMENT
```

from a Proc.

| PRIMARY INPUT | PRIMARY OUTPUT |
|---|---|
| Anything we type goes in here | The command to be processed |
| **SECONDARY INPUT** | **SECONDARY OUTPUT** |
| Errors from the command just processed | Data for the command being processed |

**Fig. 11.1.** The PROC buffers.

Since we wish to execute this as a single word command, the Proc must be placed in the master dictionary. We shall give the name REPORT to the record placed into the master dictionary which forms the Proc. The command to create this record is:

```
ED MD REPORT
New Item
Top
.
```

To indicate that this is a Proc the first line must be PQ, so we insert this:

```
.I
001 PQ
002
```

The command to be executed must now be placed into the primary output buffer. We tell the Proc that the following data is destined for the primary output buffer (POB) by prefixing the data with the letter H. It might be a good idea to annotate the Proc to ease maintenance and we can do this by prefixing comments with the letter C:

```
002 C
003 C The Access report comes next
004 C
005 HSORT PERSONNEL BY NAME WITH AGE > "25" NAME AGE DEPART-
MENT
```

Having filled the output buffer, we tell the Proc to process the contents of the output buffer by typing the letter P:

```
006 P
007
.FI
'REPORT' filed
```

We then file the Proc and we can now execute it by typing REPORT. We could have written line 5 in several stages as follows:

```
005 HSORT PERSONNEL
006 H BY NAME
007 H WITH AGE > "25"
008 H NAME AGE DEPARTMENT
```

Each successive H command adds to the end of the POB. Note that each line except the first has a space at the beginning. If we had not had a space, the contents of the POB at process time would have been:

SORT PERSONNELBY NAMEWITH AGE > "25"NAME AGE DEPARTMENT

and the Access processor would have aborted with the error message

**PERSONNELBY IS NOT A FILENAME.**

Suppose now that the process we wish to execute is an Access SELECT, the results of which are to be pipelined into a BASIC program, and the BASIC program will prompt for the file name on which we are working.

Clearly we can put the select into the POB and execute that, but how do we prevent the BASIC program name from just being added onto the end of the POB? The answer is that we place the subsequent instructions into the secondary output buffer (SOB). Data is placed into the SOB via the H command as for the POB, but first we must tell the Proc that following statements are destined for the secondary, and not the primary output buffer, by turning on the 'stack'. The SOB represents stacked input for the process which is to be executed, so the file name can be placed here too. A carriage return is indicated by suffixing a left chevron (<) to the H command. Thus this Proc is as follows:

```
001 PQ
002 C
003 C An example of the use of the secondary output buffer
004 C
```

```
005 HSSELECT PERSONNEL
006 H BY NAME
007 H WITH AGE > "25"
008 C
009 C Now turn on the stack with the STON command
010 C
011 STON
012 HRUN PROGFILE PROGNAME<
013 HPERSONNEL<
014 P
```

Now let us change the first problem so that the exact data being displayed is not specified. We are going to tell the Proc at run-time which field is to be displayed by using the syntax REPORT NAME. Thus the field to be displayed comes into the Proc in the primary input buffer (PIB) and needs to be moved to the correct place in the POB before the report is executed. We need to be able to point at the right place in the PIB and move only the second word of the command.

Proc maintains pointers to enable us to do this and spaces are used to break up the buffers so that they are in elements. With our TCL command REPORT NAME, the word REPORT is in element 1 of the PIB and the word NAME is in element 2. When we move data from one buffer to another, we move only one element. However, it is important to realise that the data moved is relative to the pointer, so we have to control the positioning of the pointer. The S command will position the pointer and the A command will move data from the PIB to the POB.

The first Proc can now be rewritten to read:

```
001 PQ
002 C
003 C A general purpose PERSONNEL report invoked by REPORT fieldname
004 C
005 C The next command positions the input buffer pointer
006 C at the beginning of the PIB
007 S1
008 C
009 C Now the main body of the report
010 HSORT PERSONNEL
011 H BY NAME
012 H WITH AGE > "25"
013 C
014 C The next command moves the second element relative to the
```

```
015 C input buffer pointer to the POB
016 C
017 A2
018 P
```

Note that there was no need to put an extra space between "25" and A2 because the command A2 will automatically place a space there.

After the data has been transferred, the input buffer pointer points to the character after the data transferred. This means that if you execute another A2 command, the fourth element relative to the beginning of the PIB, and not the second, will be transferred.

The next problem is to check that a parameter has indeed been entered in the TCL command, but if it has not been, to prompt for the field name and use that. To do this we need to be able to test the PIB and branch if something has been entered. If it has not, we need to output a prompt and input the field name.

The test will be carried out using the IF command. The branch is specified by a G command, with a branch to a statement label. The output is specified with the O command and the input will be taken with the IP command. The Proc logic before the report is amended is as follows:

```
001 PQ
002 C
003 C If the data has already been entered then OK
004 C Otherwise we will get it
005 C
006 C Here is the test
007 C
008 S1
009 IF A2 G 10
010 C
011 C Here is the prompt
012 C
013 S2
014 OPlease enter a field name
015 C
016 C Here is the input
017 C
018 IP
019 C
020 C And here is the branch destination, statement labelled 10
```

```
021 C
022 10 HSORT PERSONNEL
.....etc.
```

The next extension to the problem is that we wish to validate the input. We will prompt for the age to be used in the selection criteria of the Access sentence. This is achieved by comparing the input against a pattern match. After the IP command we would write:

```
019 S1
020 IF A2 # (2N) G 1
```

that is, if the second element of the PIB does not match two numbers, then branch to statement label 1. This would be at the Proc line which outputs the prompt. IF can test for allowed values, for instance IF A2 = FRED G 1, or forbidden values, IF A2 # FRED G 1, or ranges, IF A2 > 100 G 1. Note that when testing for specific string values, there is no need to wrap the string in quotes as with BASIC. The second statement can be any Proc command, even another IF, so we might construct a test such as this:

```
IF A2 > 100 IF A2 < 1000 OIn range
```

[ is used to represent less than or equal to and ] is used for greater than or equal to.

A powerful extension to the IF command is the ability to test for error conditions. Suppose we have a program which inputs a file name and then tries to open the file. If the file cannot be opened, the program might execute a STOP 201 and use the system error message to report that the file cannot be opened. When control is passed back to the Proc, further processing may depend on the success of the previous process. The ability to trap for the error condition is thus very useful and is carried out like this:

```
IF E = 201 G 5
```

A variation of this is a test for a select list operation. If the result of a GET-LIST or SELECT resulted in an error condition, such as the list not exiting on file or NO ITEMS PRESENT, there will be no select to drive the next process. Normally the next process is in stacked input and the Proc does not regain control after the execution of the SELECT to test for the error. However we may stack a null to force the Proc to regain control and then test for a successful SELECT. Consider the following Proc:

```
001 PQ
002 HGET-LIST FRED
003 STON
004 C
005 C Stack a null
006 H<
007 P
008 C
009 C Now test that GET-LIST was okay
010 IF # S G 10
011 C
012 C If you got to here it was okay
013 HRUN PROGFILE PROGNAME
014 P
015 10 Continue
```

These examples of Procs are getting a little too long to keep them in the master dictionary. Long items should not be maintained in the master dictionary because this increases the chance of overflow in the master dictionary. If the master dictionary is in overflow, it will take longer to open any file whose definition item is in the overflow area, or execute any process whose verb definition is in the overflow area. Master dictionaries in overflow have quite serious implications for general system response times.

To keep the Procs short, and yet retain the functionality, we put them outside the master dictionary. Master dictionary Procs should contain only two lines, PQ and then an instruction to transfer to the real procfile.

```
001 PQ
002 (EG-PROCLIB EXAMPLE)
```

This Proc simply transfers control to the Proc EXAMPLE in the file EG-PROCLIB. If the Proc is called EXAMPLE in the master dictionary, it is only necessary to place the file name in parenthesis. The Proc processor will default to the Proc with the same name as this one. No parameter passing facilities are required because the Proc buffers are global and remain unchanged on passing from Proc to Proc. We could also call the second Proc as a subroutine by putting the file and Proc name in square brackets:

```
001 PQ
002 [EG-PROCLIB]
```

Again the Proc processor will default the transfer to the Proc in EG–PROCLIB with the same name as the calling Proc. Subroutine facilities are useful where a long batch process is being put together. A set of explicitly named subroutine calls is much more readable, and hence maintainable, than a single level main line Proc which executes reports, archives, analysis programs and so on. Note that any subroutine from which you intend to return must have an exit statement X (RTN on McDonnell Douglas systems, where the command X will exit to TCL).

The syntax for an internal subroutine call is opened and closed square brackets followed by a statement label. Processing continues from that label onwards until an exit command is encountered, when control passes back to the calling point. For example:

```
010 Call subroutine at statement 100
011 C
012 [] 100
```

This idea may be extended to external subroutines, so [EG–PROC-LIB EXAMPLE] 100 would transfer control to statement 100 of the Proc EXAMPLE in the file EG–PROCLIB. However this style of Proc is not clear and is very difficult to maintain.

You will very often find that Procs are useful for menus because of their ability to chain around and regain control when a process is complete. Menus are often put together with the T (terminal output) command rather than the simple 0 command. The main reason for this is that there are cursor control functions within the T command which do not exist for 0. Here is an example of a menu Proc:

```
001 PQ
002 C
003 C General purpose sales administration menu
004 C
005 Clear the screen and output heading
006 1 T (-1), "Alpha corporation", (60,0), "Order entry"
007 T (10,2),"1      Update the order book"
008 T (10,3),"2.     Enter goods despatched"
009 T (10,4),"3.     Enter returns"
010 T (10,5),"4.     Log off"
011 2 T (10,10),"Please enter an option "
012 C
013 C Now take the input and branch on reply
014 C
015 S1
```

```
016 IP
017 IF A = 1 G 10
018 IF A = 2 G 20
019 IF A = 3 G 30
020 IF A = 4 G 40
021 G 2
022 C
023 C Update the order book
024 C
025 10 [OR-PROCLIB UPDATE-ORDERS]
026 G 1
027 C
028 Continued.....
```

The T command can therefore be used to output cursor control and cursor positioning like the BASIC @ function, or to output data, enclosed in double quotation marks, like the PRINT statement. Concatenation is specified by the commas, but the T command never automatically appends a carriage return to the end of the output. If this is required, each T command must be followed by an 0. There are also four output commands which may be specified with T. B, or any word beginning with B, such as Bell, will ring the terminal bell, C or Clear will clear the screen. I followed by a number will output the ASCII character represented by the number, and X followed by a two-digit hexadecimal number will output the relevant ASCII character calculated in hex.

## OTHER PROC COMMANDS

There are a number of other Proc commands for manipulating buffers and moving pointers. These are summarised in appendix 4 and are described in full in the Pick Reference Manual. The commands not described above should be used only rarely. It is possible to develop whole systems using Proc, but this is not recommended. A golden rule with Procs is 'keep it simple'. Complex Procs are easy to write and a nightmare to maintain.

"Pick like" computer systems invariably have major differences when it comes to Proc. The Pick Proc language corresponds to McDonnell Douglas's "Old Proc". These are still supported, even on the newest versions of McDonnell Douglas equipment. Users wishing to maintain compatability with Pick should therefore only use Old Proc on McDonnell Douglas. The McDonnell Douglas New Proc has a number of enhancements over Old Proc.

New Procs are distinguished from Old Procs in that the first line of the Proc is PQN, not PQ. Almost all the Old Proc commands are supported, but it is not possible to call a PQ type Proc from a PQN type Proc or vice versa.

Some commands are prefixed by N — IP becomes NIP, and H becomes NH, for instance. Some commands have changed their meaning — F and B do not affect the positioning of the stack pointer, as in Pick. Instead, they are GO-type commands, respectively saying go forwards, or backwards to the next Proc marker, indicated by an M. The whole area of moving data from one buffer to another has been enhanced by the addition of a MV command. Within this the input and output buffer elements may be directly referenced as 'variables'. %1 represents the first element of the currently active input buffer, %2 the second, and so on. #1 represents the first element of the currently active output buffer. Thus the command:

```
MV #1 %1
```

will move the first element of the input buffer to the first element of the output buffer. The output buffer is processed in the normal way by the execution of a P command.

Perhaps the biggest area of enhancement is the addition of file I/O facilities, the file buffer and the select buffer.

In the same way that the input buffer elements can be represented by %n, there is a method of accessing the file buffer. &1.1 represents attribute 1 of a record in file buffer area 1. &1.2 represents attribute 2 and so on. The select buffer is referenced as !1 or !2. So MV &1.1 #1 will move the first element of the output buffer to attribute 1 of the record in the first area of the file buffer.

This allows us to retrieve and update information on the database. A class of commands is provided specifically for file handling. The commands are F–OPEN, F–READ and F–WRITE, they correspond roughly to the OPEN, READ and WRITE commands in BASIC. F–OPEN and F–READ must be followed by a statement which tells the Proc what to do if the file (or record) cannot be opened (or read). F–OPEN 1 SALES would open the SALES file to file buffer area 1. Any subsequent read into this area or write from this area will operate on the SALES file.

F–READ 1 %1 would read the record with a key described by the first element of the PIB into file buffer area 1. To write this back to the file you simply write F–WRITE 1.

There are ten file buffer areas, so up to ten files may be opened at once. If more than one record from the same file is to be manipulated at once, separate buffer areas must be opened for each record. Note that there is no record locking facility.

The new Proc file I/O facilities are useful for quick checks on the status of system or global application data. It is not recommended that extensive file update routines are written in Proc. The cryptic nature of the Proc language makes maintenance difficult. All Procs, whether they be to Pick standard or McDonnell Douglas New Proc standard, should be kept as short as possible for this reason. Procs should be written in manageable modules. A module should either carry out a single task, such as an Access select followed by a BASIC program, or should be a list of other modules or tasks, like a menu or a batch processing sequence.

Prime Information and Revelation do not have any Proc language. Both have the concept of a 'paragraph' which is simply a list of statements that could have ben entered at TCL, contained in a record. Both also have an extra statement in the BASIC language which lends interesting possibilities to these systems. The EXECUTE and PERFORM statements can be made to execute any statement from BASIC that could have otherwise only been executed from TCL. These are very like the CHAIN statement that was discussed in the chapter on BASIC, with one important difference. When the initiated process has been completed control passes back to the BASIC program at the statement after the EXECUTE. This means that any system function may be called as a subroutine directly from BASIC.

# Chapter 12
# Pick's System Files

When a computer is delivered with the Pick operating system the supplier will also deliver a 'basic' system. This basic system consists of the operating system itself and at least one account.

This account is called SYSPROG and is the account where system administration functions are carried out such as creating other accounts, archiving, restarting the spooler and so on.

SYSPROG contains a number of files which contain information used by various of the Pick utilities. In general, it is not a good idea to create new files in the SYSPROG account since these will be lost in subsequent upgrades to the operating system which will be delivered with a new SYSPROG.

### SYSTEM

One of the files to be found in SYSPROG is called SYSTEM. This is the system dictionary and is the only file on the computer which is in a fixed place on the disk. As discussed in the chapter on the Pick database, SYSTEM contains the names of the various accounts available on the computer, including SYSPROG itself. By editing this the system administrator may change update and retrieval keys, privilege levels and account justification. The other fields in these records must not be amended.

To get a list of all the accounts making up the database, we can use the Access command:

```
SORT ONLY SYSTEM
```

One of the records held on SYSTEM is called LOGON. This does not define an account name, but contains the logon message in ERRMSG format (see below). Other records do not define accounts but contain Q pointers. These are synonym, or alternative, logon names for existing accounts. An example of this type, delivered with the basic system, is the Q pointer COLDSTART which is an alternative logon

name for SYSPROG. An examination of the MD entry COLDSTART will reveal the Proc executed when the system is booted. Obviously, it is possible to enter the SYSPROG account by typing LOGTO COLDSTART but only at the expense of executing the final part of the boot procedure.

### ERRMSG

ERRMSG contains all the system error messages. If garbage is entered at TCL the system responds with the error message:

```
[3] VERB?
```

This is held as a record on the ERRMSG file with an item–id of 3. Since the error messages are ordinary records on an ordinary file they may be amended with the editor to suit individual tastes.

In fact, the ERRMSG records act like a very simple command language. Each attribute of an ERRMSG record begins with a command letter which is used by the error message handler to manipulate the rest of that attribute. ERRMSG 3 has a single attribute like this:

```
E VERB?
```

The E is recognised by the error message handler and means 'output the error message number in square brackets followed by whatever text follows'.

An A as the first character inserts a parameter into the error message. This can be used to illustrate how the ERRMSG file can be used in conjunction with BASIC. ERRMSG 201 looks like this:

```
201
001 E '
002 A
003 H ' IS NOT A FILENAME
```

If the BASIC statement STOP 201,"INVOICES" is executed, error message 201 will be output as the program halts and the word INVOICES will be used as a parameter resulting in the error message:

```
[201] 'INVOICES' IS NOT A FILENAME.
```

Several parameters may be passed in this way, separated by commas, each A in the ERRMSG record taking the next parameter from the list. If the command character is X, then a parameter will be skipped without being output.

If the A is followed by a number in brackets, these parameters will be formatted left justified. A(10) would therefore take a parameter and left justify in a field of ten spaces. Right justification can be specified by a command of R, rather than A.

If the command character is H, whatever text follows the H will be output. The error message number will not prefix this message, nor will a carriage return be output at the end. To get a carriage return the command character L is used.

An individual D will output the current date, T will output the current time and S(20) will output 20 spaces.

The verb PRINT-ERR can be used to display the ERRMSG records as they will be output, so if you wish to add messages into the ERRMSG file for use with BASIC these can be tested with the PRINT-ERR verb.

```
PRINT-ERR ERRMSG 201
```

results in:

**[201] 'A' IS NOT A FILENAME.**

Error message 335 contains the message displayed when the user logs on to an account. A nice touch is to edit your company's name in here. Error message 336 is the log off message.

## BLOCK-CONVERT

The BLOCK-CONVERT file contains the character definitions for use with the BLOCK-PRINT verb.

The key for the records in BLOCK-CONVERT is the character being defined. The first field is a number which defines the width of the block character.

Subsequent fields either begin with a B (output spaces or blanks), or C (output the character). Switches between blanks and characters are specified by commas (value mark on McDonnell Douglas). B1,5,1 therefore outputs a space followed by five of the character followed by a final space.

The BLOCK-PRINT verb is used to print enlarged messages on a terminal or printer. The format used is:

```
BLOCK-PRINT message
```

BLOCK-PRINT HELLO will result in the output shown in Fig. 12.1.

If the BLOCK-PRINT command is followed by a P option, (P), the output will be directed to the sytem printer. BLOCK-PRINT is useful

```
HH    HH  EEEEEE  LL      LL       000000
HH    HH  EE      LL      LL     00     00
HH    HH  EE      LL      LL     00     00
HHHHHHHH  EEEEE   LL      LL     00     00
HH    HH  EE      LL      LL     00     00
HH    HH  EE      LL      LL     00     00
HH    HH  EEEEEE  LLLLLL  LLLLLL   000000
```

**Fig. 12.1.** The output produced by BLOCK–PRINT HELLO.

on systems where a lot of reports are produced on a single printer. The BLOCK–PRINT output makes the division between reports obvious, which enables the reports to be separated easily.

## ACC

The ACC file contains details of the lines that are currently in use. It is also the file which stores the logon details for accounts which update the accounting file. (See creating accounts.)

The LISTU verb uses the information stored in the ACC file to give a report showing the logon details for each user. The instruction:

```
LISTU
```

will result in a report similar to the one shown in Fig. 12.2.

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

*00 0200 SEMINAR     19:54  07 MAR 85 Demonstration area
 01 0220 SYSPROG     15:34  07 MAR 85 Development office
 03 0260 ADMIN       10:09  07 MAR 85 Employment office
 04 0280 SALES       09:25  07 MAR 85 Order clerk
```

**Fig. 12.2.** The output produced by LISTU.

In this report CH# is the line number that the user's terminal is attached to. PCBF indicates the disk frame address where the user's workspace begins. NAME is the account that the user is logged on to. TIME and DATE are the time and date at which the logon occurred. LOCATION represents the physical location of the terminal. This is held in the dictionary of the ACC file and can be maintained by the system administrator. The location is changed using the system editor.

In the above example attribute 1 of the record 01 in DICT ACC says "Development office", so if the terminal is ever physically moved, this record should be amended.

The historical information generated by the logon process can be displayed by typing the Access command:

```
LIST ACC WITH DATE
```

This might result in a report such as Fig. 12.3.

In this report, the column headed ACC shows the account which was

```
PAGE   1

ACC..... DATE. TIME... CONN...    UNITS PAGES
               *       *       *       *
PERSONNEL#4 08/02   14:09   00:00       2
            10/02   15:33   01:42  164654   101
            11/02   09:52   00:26  166650     4
                    10:26   02:51  165131     1
                    14:32   01:05  166447
            15/02   08:44   00:02  167734
            16/02   13:22   00:01  167669

END OF LIST
```

**Fig. 12.3.** The output produced by LIST ACC WITH DATE.

logged onto and the line number where the logon took place. DATE and TIME are the date and time when the logon occurred. CONN is the elapsed time in hours and minutes that the user spent in the account. UNITS are the number of CPU milliseconds that were used. This is useful where a user is to be charged for the use of the computer. The TCL command CHARGES may be used at any time to show the CPU millisecond usage at any time. The column headed PAGES shows the number of pages of output that were processed through the spooler.

Where accounts are set up to record this historical information, a record will be created for each line logging on to those accounts. However, there is no automatic mechanism for clearing out this information when it has been finished with. What happens is that the ACC file just grows and grows, so it should be a matter of routine that the ACC file is periodically cleared, using the CLEAR–FILE command.

## SYSPROG–PL AND PROCLIB

SYSPROG–PL and PROCLIB contain a number of programs and Procs which have been supplied by Pick Systems and/or the manufacturer. The PASSWORD utility for changing the account passwords, the LISTU Proc and the ACCOUNT–SAVE Proc are examples of these. Usually PROCLIB is supplied as a system level file, so it looks as if it is an account. However it does not have any of the usual master dictionary definitions, so if you try to logto PROCLIB you will probably see the following error message:

**ERRMSG [ERRMSG] 340 0 12765 0**

The only thing that you can do at this point is hit the BREAK key and type OFF. In general do not attempt to log to PROCLIB, ERRMSG, SYSTEM, ACC or BLOCK–TERM as these are all system level files. These should be password protected and the password should be forgotten.

## NEWAC

NEWAC contains all the verbs, program, Proc and file definitions which are used to set up new accounts. The CREATE–ACCOUNT utility copies all the records from NEWAC into a new master dictionary when an account is created. If you wish to prevent certain verbs from being in all accounts, deleting the definition from NEWAC will ensure that the verbs are never set up in the first place. On the other hand, if you have developed an application general to all accounts, new accounts can be automatically set up to have access to the application by copying the relevant program, Proc and file definitions into NEWAC.

## OTHER FILES

There may be other files delivered within SYSPROG. There may be a file called SYSTEM–OBJECT which contains the object code for the operating system and a file called JET–MODES which contains the object code for the JET word processor. These should never be amended. If they are amended, and the object code is reloaded, the system will probably crash.

Some manufacturers supply a table of terminal drivers, called CURSOR. Many manufacturers supply free utilities for graphics, spread-

sheets and/or application generation, which may be delivered as separate accounts or other files within SYSPROG.

In general, if you do wish to amend or add to any of the system files, such as BLOCK–CONVERT or ERRMSG, you have to be prepared to lose the changes if the operating system gets upgraded. Otherwise, you must make provision for your changes to be saved. For this reason it is not a good idea to create your own files or develop your own programs in SYSPROG.

# Chapter 13
# Other Pick Commands

This chapter provides a description of the use of some important Pick commands which do not fit easily into the other sections of this book.

## FILE HANDLING

Although CREATE–ACCOUNT and CREATE–FILE have been described at length, there are other verbs which deal with the clearing and deletion of data.

DELETE–ACCOUNT will delete a whole account, complete with its data files. It is a verb which is only available in SYSPROG. A listing of all the files which will be deleted is given, and the system administrator must confirm that the account is to be deleted before the process is carried out. All the disk space that was occupied by the data in the account is returned to the overflow table. It is not permissible to delete an account while a FILE–SAVE is being carried out. It is advisable that all users should be logged off when this takes place.

DELETE–FILE will delete a single data file and its associated dictionary. The format is:

```
DELETE-FILE filename .
```

CLEAR–FILE will clear either the dictionary or data portion of a file. The file still exists and may be accessed, but no records will remain. The file is 'empty'. DICT or DATA must be specified in the command:

```
CLEAR-FILE DATA filename
```

The COPY command allows data to be copied, within the same file or across files. The format is:

```
COPY filename itemlist
```

and the system responds:

**TO:**

where a new item list or file name may be specified. If the copying is to a different file, the file name must be prefixed by a bracket, for instance:

```
To:(newfilename
```

COPY can be followed by a number of options which: (a) allow for the records to be deleted as they are copied; (b) allow any existing records to be overwritten; or (c) allow records to be copied to the terminal, printer or tape, rather than another file.

COPY may be driven by an Access SELECT, rather than a specific item list. It is extremely useful for updating 'grandfather, father, son' file systems.

## TERMINAL CHARACTERISTICS

Pick has two commands which allow terminal characteristics to be set. These deal with the width and depth of the terminal and system printers, the type of terminal and specific functions, such as line feed delay and the back space character. SET–TERM, a SYSPROG only verb, sets the terminal characteristics for all users. TERM sets the terminal characteristics for a specific user. TERM is overridden by SET–TERM when the user logs across accounts. If

```
TERM
```

is entered the current settings are displayed, as shown in Fig. 13.1:

| | TERMINAL | PRINTER |
|---|---|---|
| **PAGE WIDTH:** | **79** | **80** |
| **PAGE DEPTH:** | **23** | **60** |
| **LINE SKIP:** | **0** | |
| **LF DELAY:** | **2** | |
| **FF DELAY:** | **2** | |
| **BACKSPACE:** | **8** | |
| **TERM TYPE:** | **R** | |

**Fig. 13.1.** The output produced by TERM.

To change any of these we type TERM followed by the parameters separated by commas. For example, to change the printer width to 132, leave everything else unchanged:

```
TERM ,,,,,132
```

The terminal type parameter allows different terminals to be used on the same system. If the terminal driver is present (your system supplier will advise on this), all of the cursor control and report formatting functions will work correctly after an alteration to the TERM setting.

Baud rates of terminals are changed using the SET-BAUD command. SET-BAUD 9600 will set the baud rate to 9600 at the computer end for this line. Hopefully, you can then manually change the terminal to match!

## 'CROSS TERMINAL' FUNCTIONS

Pick provides facilities for sending messages to other terminals on the same computer, and logging on and logging off other terminals.

```
MSG * All users please log off.
```

would send the message "All users please log off." to all users.

```
MSG ADMIN Please telephone computer manager.
```

would send the message "Please telephone computer manager." to any line logged on to the ADMIN account. The message is immediately relayed onto the user's screen, ringing the terminal bell in the process so as to get attention.

**15:06:28 08 MAR 1985 FROM SYSPROG:**
**Please telephone computer manager.**

The disadvantage of this is that any formatted screen being used at that moment will be spoilt.

LOGON will log another terminal on to a specific account. It is a SYSPROG only verb and will only operate if the line is currently logged off. This is useful where you wish to start a 'phantom' job. An account can be set up with a logon Proc which carries out the required task. Logging on a spare line, or an unused terminal to this account will initiate the process. This only works on computers which do not require data terminal ready (DTR) on the serial interface.

If you are logging lines on, without having a physical device attached, then there is a need for a way to log these off. The verb LOGOFF does this. Note that if you ever execute LOGOFF accidentally, when meaning OFF for instance, and just press return when prompted for the line number, terminal zero will be logged off. The best way out of this is to press the BREAK key and type END or OFF.

**RUNOFF**

Runoff is a straightforward text processing system. Many applications have documentation supplied in Runoff format but today most users prefer to produce documents with one or other of the proprietary word processing systems and so Runoff has less importance than it used to. The Pick Reference Manual has the full details of how to set up documents using Runoff. When set up, documents may be output using the RUNOFF verb. The verb format is:

```
RUNOFF filename itemlist
```

the option P may be used to output the document to the system printer. If you have a printer which only has upper case letters, such as a barrel line printer, the option U may be used to output the document wholly in upper case.

## STARTING THE PICK SYSTEM

Whenever a computer running the Pick operating system is switched on, the terminal attached to port 0 is assumed to be a system console. Thus, that terminal must also be switched on.

The power up procedure is different depending on the manufacturer and so will not be covered here. Somehow the system has to be 'bootstrapped'. On the small multi-user microcomputers this is achieved simply by switching on. On larger systems a procedure of manipulating front panel switches has to be carried out.

When the system has been bootstrapped, the option message will be sent to the system console.

```
OPTIONS (X,F)=
```

and the system will be prompting for X or F.

A reply of X will coldstart the system, i.e. the system will be made ready for use and will utilise the data already stored on the hard disk. The spooler will be started and workspace will be assigned to each port.

A reply of F will restore data from the tape device. The spooler will be started and workspace will be assigned to each port as for the coldstart but a complete file restore will take place. Meanwhile any file reorganisation will take place. All the data currently on the hard disk will be overwritten or lost. This procedure will normally be carried out whenever upgrades or some type of maintenance are carried out, or as a file reorganisation exercise.

When either of these procedures is complete the system will automatically log on to the COLDSTART account. This is a synonym account to SYSPROG but the logon Proc COLDSTART will be executed. This will probably set the time and date on systems which do not have a battery backed clock, and then verify the system modes. The verification stage calculates check sums on all the assembler modes which make up the Pick operating system and compares the result with a prestored list of check sum results. Any mismatches are reported and generally indicate that there is corruption in the ABS section of the disk. If this happens, the system should be bootstrapped from a backup device.

The printer will then be started on the appropriate line(s) and the default terminal characteristics for your system will be set.

The coldstart procedure will terminate by logging off the system console and sending the LOGON message to every line on the system. The system console now has no further significance and is free for use as a normal terminal.

Any user now switching on his terminal will receive the logon message:

**LOGON PLEASE:**

and is invited to type in the name of his account.

The process of verifying that the operating system is working correctly may be carried out from TCL in SYSPROG. It is possible that hardware errors could cause the ABS area to become corrupt. The VERIFY–SYSTEM verb verifies that it is not. Suppliers will often ask that a VERIFY–SYSTEM is carried out if you are experiencing recurring problems, such as operating system errors like CROSSING FRAME LIMITS or FORWARD LINK ZERO.

# Chapter 14
# The History and Future of Pick

Pick has been around for a long time, a very long time in terms of computer industry development. Twenty years ago two gentlemen, Don Nelson and Dick Pick, designed a "General Information Retrieval Language System" (GIRLS) for the company TRW on a US army project — a data storage and retrieval system to be used in conjunction with the Cheyenne helicopter.

This software, re-named General Information Management, or GIM, was delivered to the army in 1969 and was implemented on an IBM mainframe. GIM incorporated some of the basic features of what we now know as Pick. It had database features and an English language enquiry language. An updated version was still being operated by the Central Intelligence Agency in 1981 and may well be in use even today.

No real commercial use was made of GIM, so when the project was completed, Dick Pick was able to research his own work, which was deemed to be public domain because it had been developed under the auspices of the US Department of Defense.

The first commercial implementation of the operating system was on the Microdata[1] 1600 8-bit CPU. Pick, although still not known as such, had its own theoretical instruction set. However to implement it required something a little more sophisticated than the Microdata chip could supply, and a French company, Intertechnique, was engaged to manufacture and supply a firmware board which could interpret the instruction set, now called REAL.

Thus in 1973 the Reality operating system was born. Microdata began to market their new product by appointing dealers throughout the world. Intertechnique marketed it as Realitie on mainland Europe and a company called Computer Machinery Company (CMC) were the British dealers.

The system had obtained a medium amount of success in the years 1973 to 1976 and about a thousand end user implementations were

1. The name Microdata was changed to McDonnell Douglas Computer Systems in late 1984.

obtained. CMC were taken over by Microdata and Microdata in their turn were taken over by McDonnell Douglas. CMC were very successful in selling to large corporations in the UK who were quick to realise the value of such a flexible computer system. Dick Pick, however, became disenchanted with his continued involvement with Microdata and he left to found his own company, Pick and Associates.

Pick and Associates began to import computers from Intertechnique, implemented the operating system and sold it as Evolution. They also implemented Pick on the Honeywell Level 6 minicomputers for a company called the Ultimate Corporation. A royalties lawsuit with Microdata began which went on for several years. It ended with both parties retaining rights to the operating system. Microdata and Dick Pick were reconciled early in 1984. Ultimate subsequently purchased their rights to the Pick operating system outright and implemented Pick on the DEC LSI 11 to fill out the bottom end of their range. Dick Pick sold out his holding in Pick and Associates to Evolution Computer Systems who were subsequently taken over by Applied Technology Ventures Inc.

Around 1980 Pick developed a technique which would transfer programs written in the REAL assembler language into any given chip instruction set. This made "software" implementations possible like those implemented using 68000 and 8086 technology. This meant that special boards were no longer necessary and speeded up new implementations.

## THE "LOOKALIKES"

Around 1979 Pick "lookalikes" began to emerge. Devcom produced the Information System running on the Prime 50 series of computers. This appeared to be a powerful option. Its disadvantage, however, was that it was implemented on top of the Primos operating system and required a large amount of central memory to run effectively. In its favour it did have excellent communications facilities, which the others did not have. More recently, Cosmos implemented the Revelation system with PC/DOS on the IBM PC which has been described as "the best (database management system) available".

Subsequently Dick Pick, in his new company, Pick Systems, managed to surround himself with excellent marketing staff. They obtained commissions from many manufacturers to implement the Pick Operating System on their equipment to enable them to offer it as an option.

Pick Systems always control the implementation of Pick on a new

device. This contrasts with the implementation policy for Unix systems which have been carried out by many institutions resulting in many flavours of Unix, e.g. Unix System V, Berkeley Unix, Xenix etc., whereas there is only one Pick. This has not stopped licensees who own their implementations, i.e. Microdata and Ultimate, from developing their systems away from the mainstream Pick, albeit only slightly.

There are an increasing number of manufacturers and distributors who are more than pleased to say that their computer runs the Pick operating system. These range from the Reality through 16-bit microcomputers such as the Fujitsu 2000 to 32-bit systems such as the C.Itoh 680 range to large minicomputers like the Microdata Sequoia. Another way of looking at this is by number of users. There are single-user systems, multi-user microcomputer systems and small and large minicomputers. The IBM 4300 implementation is already able to handle about 100 users. Pick Systems estimate that the installed user base of authorised Pick computers is over 30,000. This does not include the look-alikes such as Revelation which is reported to have shipped 20,000 by the beginning of 1985 and is delivering hundreds of copies every month. Pick Systems intend to double the number of Pick computers installed during 1985. If we remember that Pick is inherently a multi-user system, there are probably in excess of 300,000 terminals with access to a computer running the Pick operating system.

## THE FUTURE

Pick Systems have already announced that the real Pick operating system (as opposed to Revelation) is available for the IBM PC-XT. This represents another direction in marketing for them. For the first time Pick Systems themselves, rather than a manufacturer or distributor, are to be the direct vendors. In the United States, Pick on the XT will sell for around half of the expected Unix price. Buyers will be able to purchase the software separate from the hardware. These factors may well combine to give Pick a long awaited boost both in number of users and in the general awareness of the computer industry.

Amongst the newer licensees for the Pick operating system are WICAT of Australia, Nixdorf, Tao Engineering and the Harris Trust of Switzerland with an implementation on the Pinnacle.

Apart from spreading across hardware, Pick Systems are still developing the operating system. The next official major upgrade, initially entitled R84 but now renamed Open Architecture, will see the removal of the 32K maximum record size restriction, introduce enhanced com-

munications facilities such as X25 and SNA, provide transaction logging features, support phantom processes and introduce a C compiler, amongst other things. Many of these features have already been implemented at Pick Systems and the next stage is for the licensees to implement the enhancements on their equipment. It goes without saying that all existing software written under Pick will be upwards compatible.

Pick Systems are also working on a co-processor, code named Vulture — a 32-bit processor which will plug into a host machine running Pick and carry out most of the processing requirements, leaving the host machine to handle input and output. This will not only increase throughput but also extend the number of terminals that may be supported by a single CPU.

## THE STANDARD OPERATING SYSTEM

Having described the various facilities that make up the Pick Operating System, the enormous potential of Pick should now be clear. Although Pick is not the right answer for all applications, it is ideal for the commercial environment. Pick's database approach gives flexibility, in that *ad hoc* requirements can be responded to, independence, in that users are not totally reliant on the availability of technical experts, and efficiency, in that new applications may use the data associated with existing applications.

It is not expected that any operating system will become the 'standard' for all applications. Each operating system will have its speciality. Pick's speciality is business and administration. It is in the areas of business and administration that Pick will be the standard that others try to match.

# Appendices

## APPENDIX 1 — EDITOR COMMANDS

This is a brief explanation of all of the commands available in the system editor. Commands suffixed by * have been explained more fully in the text. All of the commands are explained in full in the Pick Reference Manual.

| | |
|---|---|
| I* | Insert data on a new line after the current line. |
| R* | Replace the whole of the current line. |
| Rn | Replace the whole of the next n lines. |
| R/a/b * | Replace the first occurrence of string a by b. |
| R//b * | Place string b at the beginning of the current line. |
| R/a/b/n * | Replace string a occurring in column n by string b. |
| R/a/b/n-m * | Replace string a occurring in columns n to m by string b. |
| RU/a/b * | Replace unconditionally all occurrences of string a by string b in the current line. |
| Rn/a/b | Replace the first occurrence of string a by string b in the next n lines. |
| RUn/a/b | Replace unconditional a by b in the next n lines. |
| Ln * | List the next n lines. |
| L"a * | Locate the next occurrence of string a. |
| Ln"a * | Locate the first occurrence in the next n lines of string a. |
| L"a"n-m | Locate the next occurrence of string a occurring between columns n to m. |
| L:a * | Locate the next line beginning with string a. |
| A | Again, repeat the last locate command. |
| Gn * | Go to line n. |
| n * | Go to line n. |
| DE * | Delete the current line. |
| DEn | Delete the next n lines. |
| F * | File current buffer. |
| FI * | File and exit record. |
| FI item * | File, renaming the record item and exit record. |
| FI(f i * | File in file f as record i and exit the record. |
| FIO(f i | File in file f as record i overwriting any existing record and exit the record. |
| FIL | File the record as a list. |
| FIK * | File the record, exit and exit any itemlist, i.e. return to TCL. |
| FILK | File the record as a list and exit any itemlist. |

| | |
|---|---|
| FIC | File the record and compile (Revelation only). |
| FD * | File delete, delete this record from filename and exit record. |
| FDK | File delete, exit record and any item-idlist, i.e. return to TCL. |
| FS * | File the record on the database but continue editing. |
| FS item | File the record naming it as item. |
| FS (f i | File the record on file f record i. |
| FSO(f i | File the record on file f record i overwriting any existing record. |
| EX * | Exit from this record leaving the record unchanged on the database. |
| EXK * | Exit from this record and kill item-idlist return to TCL. |
| EXT | Revelation version of EXK. |
| MEn"id" * | Merge n attributes from record id in this file. |
| MEn"id"m * | Merge n attributes from id beginning at attribute m. |
| MEn(f i * | Merge n attributes from file f record i. |
| X | Reverse the effect of the last I, R or DE command. |
| XF | Reverse the effect of all the changes since the last F command. |
| T | Go to the top of the record. |
| B | Go to the bottom of the record. |
| Un | Go up n lines. |
| Nn | Go to the next nth line. |
| Wn | List 22 lines up to line n. (McDonnell Douglas only). |
| ^ * | Toggle ^. ^ is a wild card character for any string. |
| TB n,n.. | Set tabs to be n,n,... etc. |
| C * | Display column number mask. |
| ? | Display current line and item id. |
| S? | Display the current size of the record. |
| Zn-m | Display only columns n to m. |
| P * | Execute prestore command 0 (set to L22 on entry to the editor). |
| Pn | Execute prestore command n. |
| Pn c | Set prestore command n to c. |

## APPENDIX 2 — BASIC COMMAND SUMMARY

This is a brief explanation of all the commands available under Pick BASIC. Commands suffixed by * are explained more fully in the text. All the commands are described in full in the Pick Reference Manual, except those suffixed with † which are sometimes omitted. Commands suffixed by § are not yet available on all implementations but should be by 1986.

| | |
|---|---|
| ABORT | Abort execution and return to TCL. |
| BREAK OFF | Turn the break key off. |
| BREAK ON | Turn the break key on. |
| CALL * | External subroutine call. |
| CASE * | Multiway IF structure. |
| CHAIN * | External chain to another process. |
| CLEAR | Clear variable table to zeroes. |
| CLEARFILE | Like verb CLEAR-FILE. |
| COMMON * | COMMON variable list. |
| CRT * † | Output to terminal. |
| DATA * | Stack data for use with CHAIN. |
| DELETE * | Delete a record from a file. |
| DIM * | Real array dimension statement. |
| ECHO OFF | Echo of input off. |
| ECHO ON | Echo of input on. |
| END * | END of structure or program. |
| EQUATE * | Equate variables or constants. |
| EXECUTE †/§ | Execute a TCL command and return to BASIC program. |
| FOOTING * | Report footing. |
| FOR,STEP,NEXT * | FOR NEXT loop. |
| GOTO | Unconditional branch. |
| GOSUB * | Internal subroutine call. |
| HEADING * | Report heading. |
| IF,THEN,ELSE * | Conditional structure. |
| INPUT * | Take input from keyboard. |
| INPUTERR | Error message from INPUTTRAP. |
| INPUTNULL | Null character for INPUTTRAP. |
| INPUTTRAP | Single character input and validate. |
| LOCATE * | Dynamic array locate element. |
| LOCK * | Semaphore lock. |
| LOOP,WHILE,REPEAT | Loop structure. |
| MAT * | Real array global assign. |
| MATREAD * | Read from file into real array. |
| MATREADU * | Read from file into real array with update lock. |
| MATWRITE * | Write from real array to file. |
| MATWRITEU | Write from real array to file but maintain update lock. |

| | |
|---|---|
| NULL | No operation. |
| ON,GOTO | Multiway branch. |
| ON,GOSUB | Multiway internal subroutine call. |
| OPEN * | Open a file. |
| PAGE | Throw a page. |
| PRECISION | Set precision of calculations. |
| PRINT * | Output. |
| PRINTER ON * | Output to printer. |
| PRINTER OFF * | Output to terminal. |
| PRINTER CLOSE * | Close print file. |
| PROCREAD * † | Read Proc input buffer. |
| PROCWRITE * † | Write to Proc input buffer. |
| PROGRAM † | PROGRAM declaration (like SUBROUTINE). |
| PROMPT | Change prompt character. |
| READ * | Read from file into dynamic array. |
| READNEXT * | Read next field from a select list. |
| READT | Read from tape. |
| READU * | Read with update lock. |
| READV * | Read variable from field on file. |
| READVU * | Read variable from field on file with update lock. |
| RELEASE * | Release all update locks. |
| REM,! or * * | Comment. |
| RETURN * | Internal or external subroutine return point. |
| RETURN TO | Internal subroutine return with branch. |
| REWIND | Rewind tape unit. |
| RQM | Release quantum (sleep for one second and give up timeslice). |
| SELECT | Select a whole file to a select variable. |
| STOP * | Halt execution. |
| SUBROUTINE * | Subroutine declaration statement. |
| UNLOCK * | Release semaphore lock. |
| WEOF | Write end of file mark to tape. |
| WRITE * | Write dynamic array to file. |
| WRITET | Write block to tape. |
| WRITEU | Write maintaining update lock. |
| WRITEV * | Write single file to file. |
| WRITEVU | Write single field to file with update lock maintained. |

## APPENDIX 3 — BASIC FUNCTION SUMMARY

This is a brief explanation of all the functions available under Pick BASIC. Functions suffixed by * are explained more fully in the text. All the commands are described in full in the Pick Reference Manual, except those suffixed with † which are sometimes omitted.

| | |
|---|---|
| @(x,y) * | Cursor control function. |
| @(−a) * | Screen control function. |
| ABS(a) | Absolute (positive) value of a. |
| ALPHA(a) | 1 if a is alphabetic, otherwise 0. |
| ASCII(a) | Returns the ASCII value of EBCDIC string a. |
| CHAR(a) | The ASCII character number a. |
| COL1() | Beginning column of last FIELD function. |
| COL2() | Ending column of last FIELD function. |
| COS(a) | Cosine of a degrees. |
| COUNT (a,b) | Number of occurrences of string b in string a. |
| DATE () | Machine date in internal format. |
| DCOUNT(a,b) † | As COUNT +1. |
| DELETE(a,b,c,d) * | Dynamic array delete function. |
| EBCDIC(a) | The EBCDIC value of ASCII string a. |
| EXP(a) | e to the power a. |
| EXTRACT (a,b,c,d) * | Dynamic array extraction. |
| FIELD(a,b,c) | Extract the cth field from string a delimited by character b. |
| ICONV(a,b) * | User exit b using data a (input conversion). |
| INDEX (a,b,c) | Column position of cth occurrence of string b in string a. |
| INSERT (a,b,c,d) * | Dynamic array insert function. |
| INT(a) | Whole number part (integer) of a. |
| LEN(a) | Length of string a. |
| LN(A) | Natural logarithm of a. |
| MOD(a,b) † | See REM. |
| NOT(a) | Logical negation of a. |
| NUM(a) | 1 if a is numeric O otherwise. |
| OCONV(a,b) * | As ICONV but output conversion. |
| PWR(a,b) | a to the power b. |
| REM (a,b) | Remainder of a/b. |
| REPLACE(a,b,c,d,e) * | Dynamic array replace function. |
| RND(a) | Random integer in range 0-a. |
| SEQ(a) | ASCII value of character a. |
| SIN(a) | Sine of a degrees. |
| SPACE(a) | a spaces. |
| SQRT(a) | Square root of a. |
| STR(a,b) | String a repeated b times. |
| SYSTEM(a) † | The value of system variable number a. (a in the range 1-10). |
| TAN(a) | Tangent of a degrees. |

| | |
|---|---|
| TIME() | Current time in internal format. |
| TIMEDATE() | Current time and date in external format. |
| TRIM(a) | Trim excess spaces from string a. |
| a[b,c] | c characters of string a beginning at the bth character. |

**ARITHMETIC OPERATORS**

| | |
|---|---|
| + | Add. |
| – | Subtract. |
| * | Multiply. |
| / | Divide. |
| ^ | Raise to the power. |
| : | Concatenate. |

**BOOLEAN OPERATORS**

| | |
|---|---|
| AND or & | Logical AND. |
| OR or % | Logical OR. |
| 1 | Logical TRUE. |
| 0 | Logical FALSE. |

**RELATIONAL OPERATORS**

| | |
|---|---|
| = or EQ | Equal to. |
| > or GT | Greater than. |
| >= or GE | Greater than or equal to. |
| < or LT | Less than. |
| <= or LE | Less than or equal to. |
| # or <> or NE | Not equal to. |

## APPENDIX 4 — PROC COMMAND SUMMARY

This is a summary of commands available with the Proc job control language. Commands suffixed by * have been discussed in the text.

| Command | Description |
|---|---|
| A* | Move data from the currently active input buffer to the currently active output buffer. |
| B | Back up the input buffer pointer by one parameter. |
| BO | Back up the output buffer pointer by one parameter. |
| C * | Comment. |
| D | Display currently active input buffer values on the terminal. |
| F | Move currently active input buffer pointer forward one parameter. |
| FO | Move currently active output buffer pointer forward one parameter. |
| G or GO | Goto statement label. |
| H * | Move data into output buffer. |
| IF * | Conditional execution of a Proc command. |
| IH | Move data into input buffer. |
| IP * | Input to currently active input buffer. |
| IS | Input to secondary input buffer. |
| IT | Input from tape to primary input buffer. |
| O * | Output data to terminal. |
| P * | Execute command held in the output buffers. |
| PH | As P but suppress all output generated by the executing process. |
| PP | As P but display command to to be executed. |
| PW | As PP but wait for carriage return before executing. |
| PX | As P but return to TCL after execution. |
| RI | Reset input buffer. |
| RO | Reset output buffer. |
| S * | Position input buffer pointer. |
| SP | Select primary input buffer. |
| SS | Select secondary input buffer. |
| STON * | Stack on, select secondary output buffer. |
| STOFF * | Stack off, select primary output buffer. |
| T * | Terminal output. |
| U | User exit. |
| X * | Return to calling Proc, or TCL if none. |
| +n | Add n to parameter in current input buffer. |
| −n | Subtract n from parameter in current input buffer. |
| (file proc) * | Chain to another Proc. |
| [] n * | Local subroutine at label n. |
| [file proc] * | External subroutine Proc call. |

## APPENDIX 5 — THE PICK COMMUNITY

The following is a list of the main hardware suppliers who together form the Pick community.

| Company | Machine | Status |
|---|---|---|
| Altos Computer Systems | Altos 586-986 3068 | Pick licensee. |
| Applied Digital Data Systems (ADDS) | ADDS Mentor series | Pick licensee. |
| Archford Computers | Pinnacle/Excalibur | Pick licensee. |
| Aston Technology | Crystal Excel | Pick licensee. |
| C.Itoh Electronics | CIE 680 series. | Pick licensee. |
| Cosmos | IBM PC and lookalikes<br>Some MS-DOS generics | "Revelation"<br>Pick lookalike.<br>(software only) |
| Datamedia Corporation | Datamedia 932<br>ICL Clan | Pick licensee. |
| Electronique Serpe Dassault | M-68000 | Pick licensee. |
| Fujitsu Espagne | 80186 | Pick licensee. |
| Fujitsu Micro-electronics Inc. | Fujitsu System 2000 | Pick licensee. |
| General Automation Inc. | GA Zebra | Pick licensee. |
| IBC Technologies Inc. | M-68010 | Pick licensee. |
| Icon Systems & Software Inc. | Sanyo MPS 0202 | Pick licensee. |
| Information Systems (CDI) | IBM Series 1<br>IBM PC 5051 | Pick licensee. |
| Intertechnique | IN-500<br>IN-5000 | Pick licensee. |
| McDonnell Douglas Information Systems | Reality<br>Sequoia/Sequel<br>Spirit | Reality O/S<br>(originally written by Dick Pick.) |
| Nixdorf Computer | Nixdorf 8890 VM | Pick licensee. |
| Pertec Computer Corporation | Pertec 4200<br>Crystal | Pick licensee. |
| Pick Systems | IBM PC-XT<br>IBM PC-AT<br>PC-386<br>Many PC lookalikes. | Pick on the PC<br>(software only) |
| Prime Computer | Prime 50 series | "Information"<br>Pick lookalike. |
| Standard Telephones and Cables (STC) Pty | M-68000<br>CCI mainframe | Pick licensee. |

| | | |
|---|---|---|
| Systems Management Inc. | IBM 30xx<br>IBM CS 9000<br>Some compatible VM mainframes. | Pick licensee. |
| Tau Engineering | TAU M-68000 | Pick licensee. |
| The Ultimate Corp. | Honeywell Level 6<br>DEC LSI-11<br>DEC-VAX<br>IBM 43xx | Pick licensee. |
| V Mark | AT&T Unix generics | "UniVerse"<br>Pick lookalike. |
| Wicat Computer Pty. | Wicat M-68000 | Pick licensee. |
| X Mark | Turbo Tower | Pick licensee. |

## APPENDIX 6 — TRADEMARKS

Within this book many references are made to words which are trademarks. These are listed below. This list is believed to be accurate. Should it be necessary, any appropriate corrections or omissions will be included in future editions.

| | |
|---|---|
| Access | Pick Systems. |
| Altos | Altos Computer Systems. |
| DEC | Digital Equipment Company |
| IBM | International Business Machines Corporation. |
| IBM PC | International Business Machines Corporation. |
| Inform | Prime Computer Inc. |
| Intel | Intel Corporation. |
| Mentor | Applied Digital Data Systems Inc. |
| Pick | Pick Systems. |
| Pick Basic | Pick Systems |
| Prime | Prime Computer Inc. |
| Prime Information | Prime Computer Inc. |
| Reality | McDonnell Douglas Computer Systems Inc. |
| Sequoia | McDonnell Douglas Computer Systems Inc. |
| Sequel | McDonnell Douglas Computer Systems Inc. |
| Revelation | Cosmos Inc. |
| TRW | TRW Corporation. |
| Ultimate | The Ultimate Corporation. |
| Unix | Bell Laboratories. |
| Vulture | Pick Systems. |
| Xenix | Microsoft Corporation. |
| Zebra | General Automation Inc. |

## GLOSSARY

| | |
|---|---|
| account | 1. A user name. 2. A collection of logically associated files. |
| algorithm | A set of rules for solving a problem. |
| array | A set of data identified by a single name. |
| ASCII | An acronym for American Standard Code for Information Interchange, a computer code for representing alphanumeric characters. |
| attribute | A field of data within a Pick record that may be further subdivided into values and sub-values. |
| attribute mark | The character which Pick uses to separate attributes, ASCII character 254. |
| BASIC | An abbreviation for "Beginners All Purpose Symbolic Instruction Code" — A high level programming language now very popular on micros. Pick BASIC has many special facilities in relation to the Pick database which make it much more powerful than ordinary BASICs. |
| bootstrap | The process of starting a computer. |
| byte | A character of data. |
| checksum | A method of data verification where a calculation is carried out on the data and the result is compared with the result, or checksum, that is already stored on the computer. |
| coldstart | The process which takes place immediately after a Pick computer has been bootstrapped. Usually this checks the integrity of the operating system, starts the system printers and sets the terminal drivers. |
| concatenate | Join together. |
| contiguous | Physically adjoining. Contiguous frames are disk frames which are physically next to each other on the disk. |
| D pointer | A record found in a dictionary which defines a file by pointing at the absolute frame address at which the file begins. |
| data | Information of any type. |
| database | A collection of data stored in an organised manner so that the data may be stored and retrieved easily. |
| default | The action taken in lieu of any other instruction. |
| dictionary | A file which holds records that define the structure of data held on a data file. |

| | |
|---|---|
| dynamic array | A data structure available in Pick BASIC which exactly reflects the structure of records on the database. |
| EBCDIC | An acronym for Extended Binary-Coded Decimal-Interchange Code. A computer code for representing alphanumeric characters. |
| field | A single piece of data, e.g. the name of a customer. |
| file | A collection of logically associated records. |
| hardware | The computer and any associated peripheral equipment. |
| hexadecimal | A system of counting in sixteens very often used by computer people because each digit in hexadecimal notation (0-9 and then A-F) can represent four binary digits (binary being a system of counting in twos which is used by computers). |
| hold file | A spooler file stored on the disk. |
| item | A data record. |
| item-id | (Item identifier). The unique piece of data which distinguishes a record in a file from all other records in the same file. Often called a record key. |
| justification | The method used to format data on output by lining up to the right or left. |
| Kbyte | 1024 characters. |
| key | 1. An item identifier. 2. (Sort key) Data to be sorted. |
| logon | The process by which a user enters a multi-user computer system. |
| logoff | The process by which a user leaves a multi-user computer system. |
| master dictionary | A file which holds records that: 1. Define all the files accessible from a particular account. 2. Define the vocabulary open to the users of a particular account. |
| modifier | In Access, a word which modifies the meaning of an Access sentence. |
| modulo | 1. The number of groups in a Pick file. 2. The number used as a divisor in the hashing algorithm which is used to retrieve a record from a Pick file. |
| operating system | Software which controls the use of the central processing unit and any peripheral devices. |
| option | In Access, a single character code surrounded by brackets which may be used instead of a modifier. |
| password | A word which allows access to an account or allows a protected process to begin. |
| port | The socket on the computer into which the wire from a user's terminal is plugged. |
| program | A sequence of precise instructions which specify an algorithm. |

| | |
|---|---|
| Q pointer | A record held in a master dicitionary which allows a file to be accessed from one account while being physically stored in another. |
| record | A logical entity in a file. e.g. The data belonging to a single customer in a file of customers data. |
| record key | Item identifier. |
| relational database | A database organised in a manner which allows access to any piece of data via a unique key and which can carry out certain tasks (Join, Project and Select). |
| separation | The number of frames of disk space allocated to a single group in a file. |
| software | The programs and routines which control the operation of a computer. |
| spool | An abbreviation for "Simultaneous Production of Output Off Line" — To store and queue output before it is sent to a printer or tape. |
| sub-value | The second sub-division of a Pick data record. |
| sub-value mark | The character that Pick uses to separate sub-values, ASCII character 252. |
| TCL | Terminal Command Level or Terminal Command Language. |
| terminal | Any input/output device used to communicate with a computer. |
| value | The first subdivision of a Pick data record. Multiple values might be used to hold the individual lines of the address of a customer. |
| value mark | The character that Pick uses to separate values, ASCII character 253. |

# Index

## PICK™ for users

*PICK™ for users* is intended as a guide for users and prospective users of the Pick operating system. It is designed to enable those without specialist knowledge to interrogate a Pick database and manage a computer running Pick. *PICK™ for users* covers the topics of access, the Pick database, the system editor, dictionaries, the spooler, security, archiving, Pick/Basic and Proc. The text is illustrated throughout with examples that will be familiar to the businessman, whether or not he is an existing user.

### *About the author*

Martin Taylor has been a user of the Pick operating system since 1979. Following work with a major Pick hardware distributor he is now the managing director of his own computer services company. He is the author of many business software packages, provides consultancy services for several large Pick users and gives training courses on various aspects of the operating system. He is the holder of a first class honours degree from Leeds University and national and international periodicals have published his articles and opinions on the Pick operating system and related fields.

## Titles of related interest

**Practical Data Communications: Modems, Networks and Protocols**
F. Jennings
1985. 250 pages, 62 illustrations
Essential reading for all interested in data communications systems, this handbook describes how communications interfaces, modems, networks and data link protocols are used to interconnect computer and terminal equipment. It covers in one volume all the different types of data communications networks currently used in the UK.

**Pragmatic Data Analysis**
R. Veryard
1984. 96 pages, 28 illustrations
This book introduces the idea of a data model as a clear way of depicting logical data structures. Techniques are described for building a data model by interviewing the future users of a planned information system. As a commonsense introduction to the concepts and techniques of data analysis, the book will be useful both to experienced data analysts and to students as a supplement to their training in a formal methodology.

BLACKWELL SCIENTIFIC PUBLICATIONS LTD
Osney Mead, Oxford OX2 0EL
8 John Street, London WC1N 2ES
23 Ainslie Place, Edinburgh EH3 6AJ
52 Beacon Street, Boston, Massachusetts 02108, USA
667 Lytton Avenue, Palo Alto, California 94301, USA
107 Barry Street, Carlton, Victoria 3053, Australia