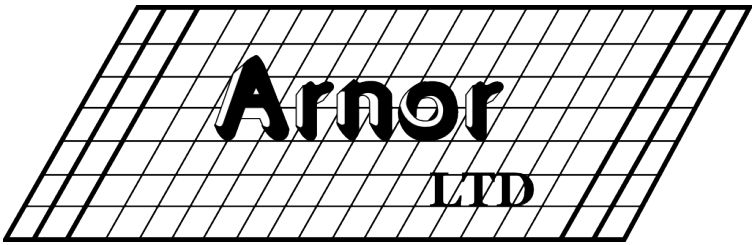
PROMERGE

PROMERGE PLUS

Disc and ROM

AMSTRAD CPC464
CPC664 & CPC6128



PROMERGE

PROMERGE - PLUS

Amstrad CPC464 CPC664 CPC6128

CONTENTS

 Issue 2 (v1.04)

The PROMERGE programs were developed using the MAXAM assembler ROM.
This manual was written on PROTEXT and PROMERGE PLUS.
The spelling was checked using PROSPELL.

MAXAM is available on ROM, disc, and cassette.
PROSPELL is available on ROM and disc.
PROTEXT is available on ROM, disc, and cassette.

Manual written by David Foster.
Arnor Ltd., 118 Whitehorse Road, Croydon, CR0 2JF.

1. INTRODUCTION

This manual describes the use of PROMERGE and PROMERGE PLUS. Both programs are intended for use in conjunction with PROTEXT and provide a very flexible and powerful mail merging facility.

PROMERGE is available only on disc and provides primarily a mail merging facility, but also includes many extensions to PROTEXT commands.

PROMERGE PLUS is available on ROM only and contains all the features of PROMERGE, but with the addition of a number of further enhancements to PROTEXT.

The manual progresses from a description of the use of PROMERGE, to the additional features of PROMERGE PLUS and finally to a detailed appendix and a number of example text files.

PROMERGE AND PROMERGE PLUS are intended ONLY for use with PROTEXT and all the facilities provided are used from within PROTEXT, either as extensions to existing commands, or as new commands. Many of the existing PROTEXT commands can now be used with optional parameters to extend their use.

Mail merging can be a complex subject, so this manual is written on the basis that the user has no knowledge of the subject and instead of listing all the facilities at once, introduces them gradually. In this way it is easy to carry out simple mail merges and, with experience, to progress to more complex examples. At the back of the manual is a technical appendix which details all the commands in alphabetical order.

Throughout the chapters on mail merging, many examples are given, with the recommendation that they be typed in, tried and saved. On occasions earlier examples will be used again. and expanded to demonstrate the use of new commands. Many of the examples can be used as 'template' files and modified at a later date to suit the needs of users.

2. GETTING STARTED

NOTE: PROMERGE and PROMERGE PLUS can only be used in conjunction with PROTEXT, which must be version 1.00 or later. Users with a version number lower than V1 .00 will require an upgraded version and should contact Arnor.

The remainder of this chapter deals with their installation and should be read carefully.

1. PROMERGE

NOTES:

(a). PROMERGE is available only on disc and will work with either disc or ROM versions of PROTEXT.

(b). It is not possible to have disc versions of PROTEXT, PROMERGE and PROSPELL all loaded into memory at the same time. Either PROMERGE or PROSPELL may be loaded with PROTEXT, but not both.

If you have the disc version of PROTEXT, load it in the usual way, then type Q to return to BASIC. If you have the ROM version of PROTEXT there is nothing to do prior to loading PROMERGE.

To load PROMERGE insert the PROMERGE disc and type RUN"DISC. The program will then load. Type |P to enter PROTEXT and then type VERSION. If PROMERGE has been loaded correctly a message will appear listing the version numbers of both PROTEXT and PROMERGE.

WARNING:

(a). Load PROMERGE before entering any text. If you load PROMERGE when there is text in memory the text will be lost.

(b). Users of the disc version should be cautious of using external commands which require the use of memory, from within PROTEXT, such as UTOPIA'S DISCCOPY command, as problems caused by overwriting part of the programs can result.

NOTE: Disc protections

The PROMERGE disc in supplied in protected form, and so may not be copied. The disc is guaranteed against failure and will be replaced for the post of a blank disc, if necessary. If an attempt is made to copy the PROMERGE disc, the message "Disc read fail" will be displayed. This is normal and is not an indication of a fault. Formatting the disc will destroy the contents. The PROMERGE disc must always be used with the 'Write protect' tab in the protected position.

2. PROMERGE PLUS

NOTE: PROMERGE PLUS is only available on ROM and can only be used with PROTEXT on ROM.

PROMERGE PLUS is supplied in a 16K EPROM which can be fitted into any available ROM expansion board. You may find that the legs of the EPROM will need straightening a little in order to fit the socket easily. Do this with extreme care by holding the EPROM in the centre (avoid touching the legs) and pressing the side of the legs against a flat surface. Keep the EPROM away from sources of electrostatic charge such as the monitor screen.

ROM numbers

Each ROM, when installed, must have a unique ROM select number so that the firmware may access it. PROMERGE PLUS is a background ROM, which means it provides facilities that can be used from other ROMs, in this case PROTEXT. The number associated with a background ROM must lie between 1 and 15 (with the V1.0 firmware of the CPC464 it must be between 1 and 7). The disc operating system is ROM 7 and if you have the Arnor AD1 or AD2 cartridge with MAXAM, PROTEXT, or UTOPIA, the cartridge will be ROM 5.

Important installation note

PROMERGE PLUS may be installed with any valid select number except 7, with the following condition:

PROTEXT must be installed, and must have a higher ROM select number than PROMERGE plus. If the prospect ROM has a lower select number or is not present at all, a warning message will be displayed when you switch on, and PROMERGE PLUS will not function.

Installation should be carried out in accordance with the ROM board instructions. Sometimes it is necessary to fit a small link to the board.

When the PROMERGE PLUS ROM is installed, switch on the machine. Enter PROTEXT with |P and type VERSION. A message should appear, listing the version numbers of PROTEXT and PROMERGE PLUS. If the message does not appear, check that a ROM select number already occupied by another ROM has not been chosen, or any links required have not been forgotten.

Note for CPC464 and CPC664 users

The CPC6128 keyboard is different from the 6PC464 and CPC664 keyboards. Wherever this manual refers to the RETURN key, it means the large ENTER key on the CPC464 and CPC664.

Conventions used in the Manual

Throughout the manual, editing commands are written in the form CTRL-Y, or CTRL-COPY. Commands used in command mode are shown in upper case, as in 'PRINT' or 'SAVE' but may actually be entered In either upper or lower case, and in many cases abbreviations may be used.

Upgrading

The disc version may be upgraded to the ROM version (PROMERGE PLUS), which contains extra commands. Upgrades are also available from disc versions of PROTEXT to the ROM version - contact Arnor for details.

The main advantages of ROM software are that the programs are always available for instant use, the full memory Is available for you to use and, in the case of PROMERGE PLUS, many extra facilities are provided.

Further software

PROSPELL. a spelling checker, is also available on ROM and disc.

3. WHAT IS MAIL MERGING?

In its simplest form, mail merging is the process of incorporating text from one file into the text of another file. One of the most commonly used forms of this is where names and addresses are read from a data file and incorporated into a standard letter, in order to create a "personalised" letter for each person. There are, however, many variations on this idea and the next few chapters will describe these, starting with simple examples and gradually introducing the more advanced features which are a part of PROMERGE and PROMERGE PLUS.

PROMERGE and PROMERGE PLUS are very advanced programs and will permit the carrying out of virtually any sort of mail merge. Selective, conditional and alternative text merges are all possible and will be covered in detail in later chapters.

Despite being very advanced programs, they have been designed in such a way that straightforward mail merging is very simple to understand and use, whilst at the same time facilities are incorporated which will permit the most extensive of mail merges and calculations.

One of the features most commonly found in connection with mail merging is the use of data files and the next chapter covers data files in detail.

NOTE: For the remainder of this manual PROMERGE PLUS will only be referred to on the occasions where it may differ from PROMERGE, or the subject is only related to PROMERGE PLUS. Unless otherwise specified., it can be taken that any references to PROMERGE are equally relevant to PROMERGE PLUS.

4. DATA FILES

Data files are a very important aspects of mail merging, so before looking at mail merging, it is important to know exactly what they are and how they are created.

A data file is a collection of items of information which may be anything from names, addresses and telephone numbers to paragraphs of text, or collections of numbers.

Data files can be created in a number of ways. The simplest example would be one created by PROTEXT, in which case the data would be entered using PROTEXT in the normal way and then saved. Most good databases and many spreadsheets are capable of creating data files suitable for use by PROMERGE. A third way is by use of the Basic OPENOUT command and PRINT#9 or WRITE#9, to process and store data from your own programs.

Any Ascii file may be used by PROMERGE as a data file, providing the data is stored in a suitable format.

Data file format

A data file will usually contain a number of records, each of which will contain one or more fields of information.

For example:- A data file might contain a list of names and addresses, in which case the name is considered as one field and each line of the address as a further field. All the fields that constitute one complete name and address are considered to be one record.

When PROMERGE reads from a data file, it reads characters from the beginning of the file, until it finds either a comma or a carriage return character, either of which it will take to indicate the end of a field. It will then take the next character it finds to be the start of the next field. Any leading spaces in a field will be ignored.

The only exception to this is when the first character of a field is found to be either a single or a double quote, in which case PROMERGE will continue to read the characters until it finds matching quote marks, which it will then take to indicate the end of that field. Any quotes of the alternate sort, or commas, contained within the matching quotes will not be treated as end of field markers in this case, but as part of the text, In addition, any leading or trailing spaces inside the quotes will be retained as part of the text.

Normally each record will be separated from the next by a blank line and this will be used by PROMERGE to indicate the end of a record.

Occasions can arise where one or more of the records may include fields which contain no information, for example when one of the fields is for telephone numbers. Unless the absent field is the last field in the record, it is vital that the field is not left out of the record and empty fields can be represented either by an empty line, or, alternatively, the dollar symbol '$' may be used to represent an empty field. When reading data, PROMERGE recognises the dollar symbol as an empty field and converts it to an empty string. One of the advantages of using the dollar symbol is that it makes it easier to look at the data file and see where each record starts and ends, but there are other reasons why it may be preferable, which will be discussed in later chapters.

Examples of valid data file records

J Smith
$
The Cottage
Little Deighton
Nr York
West Yorkshire

D Grey

15 Ambleside Road
Moortown
Harrogate
Yorks

"M Black"
"0539 412345"
' "The Leys" '
"Ulverston"
"Cumbria"

"P Brown", " ", "16 High Street", "Sandylands", "Kendal" "Cumbria"

R White, 9923 46143, 24 North Ave, Hampton, Keswick, Cumbria

J Green, $, 64 Park Road, Stockport, Cheshire, $

All the above examples are valid forms for data files suitable for use with PROMERGE and could all be used in the same data file, although it is preferable to use only one format in any data file, if only for reasons of ease of creation and alteration.

5. STORED COMMANDS

Many of the new facilities offered by PROMERGE are called into operation by the use of stored commands, so it is essential that the user is familiar with their use in PROTEXT before trying some of the advanced features described in later chapters. To this end, this chapter illustrates the subject of stored commands in a slightly different way from the PROTEXT manual and should be read in conjunction with the section on stored commands in the PROTEXT manual.

The second part of the chapter lists all the general purpose commands which are not directly connected with mail merging or printer control, detailing any new commands, or those that may be new to users of earlier versions of PROTEXT. AII stored commands are duplicated in PROMERGE, so that they are available to all users, whatever version of PROTEXT (above V 1.00), is being used. Commands connected with mail merging are covered in chapters devoted to the subject of mail merging and a separate chapter covers the many new commands concerned with control of the printer.

PROTEXT has always permitted the controlling of various of the printing parameters in two ways, firstly through the use of SETPRINT and secondly through the use of stored commands. Many people will be used to controlling these using the SETPRINT commands. SETPRINT has the effect of changing the default settings of the various print options, that is, the settings which will be used for all documents and is best considered in this way, rather than as a means of setting up individual documents.

Stored commands in PROTEXT can be used for many things, including setting page length, margin sizes, page numbering, printing text at the top or bottom of every page and line spacing. By using stored commands within a document, it is possible to set the appropriate options relating to that individual document, and these will take priority over the default settings. This is a very flexible method since different types of document may require quite different margin sixes, page lengths or line spacing, for example.

The ruler line that you use to define the width of the text is a special example of a stored command. This can be used in conjunction with the stored command SM (side margin) to ensure that text is printed centrally on the paper..

Template Files

An important concept is that of creating a number of 'template' files, arranged for commonly used document formats, which are saved on disc and loaded when required. As an example, you might have one template file for 'letter format', another for 'double spaced manuscript' and so on.

A template file would contain amongst other things, a ruler line and a number of stored commands defining the page length, the margins, continuous or single sheet stationery, etc. In some cases the template might contain some standard text. For example, an invoice template could be a complete invoice with gaps to enter the name and address, details and the figures, or a letter template could contain the sender's address and telephone number.

When a new document is started, it is only necessary to load the appropriate template file and everything will be automatically configured to suit it. Another advantage of using stored commands is that the template will be saved with the document, ensuring that when the document is next loaded, all the parameters will be as before.

Example of a 'letter template'.

```
>CO Letter template
>SM 8                    ; - Side Margin 8
>PL  72                  ; - Page length 72 (for 12 inch paper)
>CP OFF                  ; - Continuous printing off (single sheet)
>----!-----------------------------------------------------------------------------------------------------R
```

Example of a 'manuscript template'.

```
>CO Manuscript template
>TM 2                    ; - Top margin 2
>LS 2                    ; - Double line spacing
>CP ON                   ; - Continuous printing
>PN 1                    ; - Page number from 1
>HE                      TITLE OF DOCUMENT
>FO                             Page %
>----!-------!-------------------------------------------------------------------------------------------R
```

NOTE: On occasions a document may contain a considerable number of stored commands. One effect of this, which may be noticed, is that scrolling through a document may be considerably slower, as PROMERGE checks every stored command when a line is scrolled. If the PROTEXT CTRL-P option is used to disable page mode, the speed of scrolling will be increased considerably.

General purpose stored commands

The remainder of this Chapter describes the stored commands which are not directly connected with mail merging or control of the printer. New commands and commands which are not common to all versions of PROTEXT are described in detail. The PROTEXT manual should be consulted for details of commands which are mentioned but not described.

Rather than being listed in alphabetical order, the stored commands are grouped into a number of categories. A technical appendix in the back of the manual lists them in alphabetical order, with full details of syntax and usage.

NOTE: Full details of the use of variables in stored commands will be found in the chapter on 'Making the most of variables'.

NOTE: It is now possible to put comments on the same line as any stored command. The semi-colon (;) is used to indicate to PROMERGE that any further text is merely a comment and is not to be acted on.

Paper formatting

The following commands are concerned with the dimensions and positioning of the area of the paper to be printed on and include the setting of margins, page lengths etc.

*	>BM	n	Bottom margin.
*	>EM	n	Even side margin.
*	>FM	n	Footer margin.
*	>HM	n	Header margin.
*	>OM	n	Odd side margin.
*	>PL	n	Page length.
*	>SM	n	Side margin.
*	>TM	n	Top margin.
	>ZM		Zero all margins.

Descriptions of the commands prefixed with an asterisk (*) will be found in the PROTEXT manual.

ZM Zero all margins.

Description Following this command all margins will be set to zero. This command has a number of uses, one in particular being when used in conjunction with PRINTF, to create a pure Ascii file.

Note: Any margins required can be reset after the use of this command, using the appropriate stored command.

Page Formatting

These commands are concerned with how the text is presented on the area of paper defined by the previous commands and cover items like headers, footers, page numbering, line spacing and formatting.

	>CE	text	Centre line.
	>CP	ON/OFF	Continuous/single sheet printing.
	>EA	n	End at page.
*	>EF	text	Define even footer text and turn footers on.
*	>EH	text	Define even header text and turn headers on.
*	>EP	(n)	Even page throw (can be conditional).
	>FF	ON/OFF	Form feeds enabled/disabled.
*	>FO	text	Define footer text and turn footers on.
*	>FO	ON/OFF	Turn footers on/off .
	>FP	ON/OFF	Formatting whilst printing on/off.
*	>HE	text	Define header text and turn headers on.
*	>HE	ON/OFF	Turn headers on/off .
*	>LS	n	Line spacing.
	>NC	n	Number of copies.
	>NP	ON/OFF	Enable/disable new page at end of printing.
*	>OF	text	Define odd footer text and turn footers on.
*	>OH	text	Define odd header text and turn headers on.
*	>OP	(n)	Odd page throw (can be conditional) .
*	>PA	(n)	Page throw (can be conditional).
*	>PN	nn	Page number.
	>PS	ON/OFF	Page sensing on/off.
	>RJ	ON/OFF	Right justify on/off.
	>SA	n	Start at page.

Commands marked with an asterisk (*) in the list are already a part of PROTEXT and details of these commands will be found in the PROTEXT manual. Only those commands exclusive to PROMERGE or not common to all versions of PROTEXT are detailed below.

CE Centre text in line.
Description: This command is present in PROTEXT, to centre the line during re-formatting, but the use is extended to include centring during printing as well.

CP ON/OFF Continuous/single sheet printing.
Example: >CP ON
>CP OFF

Description: "ON" selects continuous printing, "OFF" selects single sheet printing. The default setting is "ON".

EA End at page.
Example: >EA number

Description: Sets the last page number to be printed. Equivalent to the corresponding option in SETPRINT.

FF ON/OFF Form feeds enabled/disabled.
Example: >FF ON
>FF OFF

Description: When enabled a form feed character (12) is printed at the end of each page. The default setting is "OFF"

FP ON/OFF Formatting whilst printing on/off.
Example: >FP ON
>FP OFF

Description: When enabled the text is formatted during the printing operation. The command '>RJ' controls whether right justification is carried out or not. Even when disabled, any paragraph that includes a variable reference will be re-formatted as a matter of course. The default setting is "OFF"

NC Number of copies.
Example: >NC number

Description: Sets the number of copies to be printed. Number must be between 1 and 255. Equivalent to the corresponding option in SETPRINT.

Note: This command is ignored if a data file is defined.

NP ON/OFF Enable/Disable new page at end of printing.
Example: >NP ON
>NP OFF
Description : When enabled the paper is moved to the top of the next page at the end of printing. When disabled the gaper is stopped as soon as the last line of text has been printed. The default setting is "ON".

PS ON/OFF Paper sensing on/off.
Example: >PS ON
>PS OFF

Description: This command only works with Epson compatible printers. When enabled the code ESC "8" is sent at top of each page, and ESC "9" at the bottom of each page. The effect of using this command is to put the printer 'off line' at the end of a page to allow the changing of single sheet paper. It is particularly useful when used with a printer buffer. The default setting is "OFF".

Note: This command does not work with the Kaga Taxan, because paper sensing only works with continuous paper on this printer.

RJ ON/OFF Right justify on/off.
Example: >RJ ON
>RJ OFF

Description: This command is used to select whether right justification is carried out, if the formatting whilst printing option '>FP' has been selected. The default setting is "ON".

SA Start at page.
Example: >SA number.

Description: Sets the first page number to be printed. Equivalent to the corresponding option in SETPRINT.

Miscellaneous commands

	>CO text	Comment line.
	>CS text	Clear screen and display message.
	>DM text	Display message.
*	>EX text	External command.
	>ST text	Stop printing and display text.
*	>WT text	Wait and display text.

CO Comment.
Example: >CO These lines do nothing except permit you
>>> to put comments and reminders in the text.

Description: This command has been extended from its use in earlier versions of PROTEXT, in as much as '>>>' is also permitted as an alternative to 'CO'.

Note: It is also possible to put comments at the end of any stored command lines, by preceding them with a semi colon (;). Anything after the semi colon will be ignored.

CS Clear screen.
Example: >CS Enter Information when requested.

Description: The screen is cleared and any message is then displayed on the Screen.

DM Display message.
Example: >DM This letter is to &name&.

Description: The message is displayed on the screen.

ST Stop printing.
Example: >ST No further text to be printed.

Description: Printing stops immediately. Any message is displayed on the screen. Further text is not printed.

6. SIMPLE MAIL MERGING

NOTE: Users of PROMERGE PLUS might find it useful at this stage to read the chapter on PROMERGE PLUS, as certain features, such as 'two file editing', will be found convenient when experimenting with the examples in this chapter.

NOTE: For the remainder of the chapters on mail merging, stored commands will be described in a simplified fashion. A fully detailed appendix, giving technical details of each command, will be found at the back of the manual.

The principle on which PROMERGE carries out mail merging is very simple. At the start of a document, PROMERGE opens a specified data file and reads the contents of each field, allocating them to variables. When all the fields for one record have been read into the specified variables, PROMERGE continues to print the document, inserting the contents of each variable into the text, whenever a variable reference is found. When the end of the text file is, reached, PROMERGE then repeats the process, reading further fields from the data file.

Variables are inserted in the text, at the appropriate position, by placing an ampersand ('&') immediately before and immediately after the variable name. For example, to insert a variable called 'name' into the text, it is simply entered as '&name&'. There is no restriction on the number of times each variable may be used in the text.

If variables are positioned in the text using tab markers, then the contents of the variable will be justified about the tab location when printed, in the normal way.

NOTE : Users of versions of PROTEXT which have the decimal tab facility should note that the contents of any variable positioned in the text using the decimal tab will automatically be positioned accordingly, enabling automatic justification of any tables of figures contained in a data file.

One of the most common forms of mail merging is where a number of letters are to be sent to individuals and their name and address is required on each letter.

Initially, there are only three stored commands that may be required, two of which will be used in the majority of cases of mail merging. These are:-

>DF	file {file}	Define data file.
>RU	var {var}	Read variables unconditionally.
>RV	var {var}	Read variables.

NOTE: The use of curly brackets indicates that optionally, one or more extra items may be specified. 'file' may be any valid filename and 'var' may be any word starting with a letter from A - Z or a - z and may contain letters, numbers, '.' and '?'

DF Define data file or files.
Example: >DF file1 file2 ... filen

Description: These files should contain data which is going to be used for mail merging. Usually there will be only one file, but several are permissible. This command is used to tell PROMERGE the name of the file to be used, and so must occur before reading data from the file.

NOTE: It is not possible to use DF and IN at the same time. If IN or more than one DF command are required a CF command must be used first to close one file before opening the next (See Advanced mail merging) .

NOTE: The drive letter may be specie fed, preceding the file name. e.g. B:file1 A:file2.

RU Read variables unconditionally from the data file.
Example: >RU var1 var2 ... varn

RV Read variables from the data file.
Example: >RV var1 var2 ... varn

'RU' and 'RV' are very important commands, being the means of extracting data from a data file for use during mail merging. Whilst both commands are used in a similar fashion, with the same syntax, there are important differences between the way the two commands work, the choice of which command to use being largely determined by the format of the data file. A thorough understanding of both these commands is important and the differences in their uses will be discussed at various stages in the manual. The number of variables named in these commands must be the same as the maximum number of fields in a data record, usually with one added for the blank line between records. They need not all be in one 'RU' or 'RV' command, but by the time a complete document has been printed once, all the f gelds for a record must have been read.

Getting started

At this stage, it would be a good idea to create a small data file with only five or six records and then enter up the example which follows and experiment with it.

At many stages throughout the manual, it will be suggested that text and data f ilea are Caved. There are two reasons for this, firstly because many of the examples will be used at a later stage and expanded as new instructions are introduced, and secondly, many of the examples will readily convert into 'template' files, which can easily be modified to create mail merge files for a variety of purposes.

The concept of keeping templates f ilea has been mentioned in an earlier chapter and is equally valid, if not more so, with mail merging documents. For many people the example files created in these chapters will be all the 'templates' they need, requiring only a few changes to suit different documents. In this way, it is possible to quickly create new documents without the need to start from the beginning each time.

Creating a Data file

The easiest way to create a small data file is simply to type it in, using PROTEXT. Enter the name on the first line and press RETURN (ENTER on the CPC464/664), then type in the address on separate lines, using RETURN at the end of each line. For the purposes of the example, make the records a mixture of three and four line addresses and for the time being, do not put any punctuation marks or quotes in the fields. After entering the last address line of each record, press RETURN a second time, so that there is a blank line between each record, and do the same at the end of the file. Save the file with the name 'DATFILE1'.

Clear the text from the computer, using PROTEXT's 'CLEAR' command and enter the following example program. Save this as 'EXAMPLE1', as it will be used in later examples and modified as further commands are introduced.

Example - simple mail merge to read names and addresses from a data file and insert them into a letter.

```
>CO     EXAMPLE1
>PL 24                  ; rem for example purposes only
>CP OFF                 ; rem single sheet
>DF datfile1            ; rem define data file
>RV name addr1 addr2 addr3 addr4 dummy          ; rem read variables.

&name&
&addr1&
&addr2&
&addr3&
&addr4&

Dear &name&,
        Thank you for your letter about the insurance for handle &addr1&
&addr2& &addr3& &addr4&. We thank that you will find the rates quoted . . . . . .
```

Before trying out the example, it is worth studying it to see what has been done. The first line is a comment line and is used as a reminder of what the document is. The 'PL 24' command in the second line and the 'CP OFF' in the next line are present only for the purposes of this example, in order that when it is printed to the screen, each record will be printed and then stop, so that it can be studied.

' DF datfile ' defines the name of the data file, so that PROMERGE can open it for reading. The following line, which in the first example makes use of the 'RV' command, is there to read the various fields of the record and allocate them to the variables. It is good practice to choose variable names that are related to the data that the variables will contain.

Special notice should be taken of the variable 'dummy', which will be used in the majority of mail merges and is used to read the empty line between each record.

'RV' has been used in this example for a special reason. One of the special characteristics of 'RV' is that when it reads from the data file, it stops reading any more data as soon as it finds an empty field and automatically gives a null value to any variables on that 'RV' command line which have not yet been allocated values.

With the data file that has been created this is important, as some of the records have less fields of data than others. If the record was one that had only three address lines and all the variables were to be read and allocated, then the variable called 'addr4' would contain the empty line between records and 'dummy' would contain the name field from the next record. Then when the next record was read, the first field would be allocated to the variable 'name', which in this case would be the first address line. By using 'RV', as soon as the blank line is read, no further lines are read and all the remaining variables are given a null value.

The remainder of the example is the text that will be printed and, as described earlier, wherever the data is required, the appropriate variable is entered in the text, wrapped in its markers.

To try out the example and save wasting paper use the PRINTS (PS) command to direct the output to the screen instead of the printer. PS is particularly useful with mail merging, as it allows you to see and check that everything is happening as expected, before printing it out properly.

One thing that will be noticed is that even though some of the addresses may have only three lines, PROMERGE closes up the blank line and does not leave unsightly gaps. The space between tie last line of the address and 'Dear . . . ' will always remain as one line .

If, for any reason, it is desired to retain the gap, PROMERGE has an alternative set of variable markers, ' ! ' to replay the ' & ' and these can be used instead. Example ' !addr4! '.

Even more noticeable is the way that the remainder of the paragraph of text containing the address, is automatically reformatted during printing, ensuring continuity of the text.

The two commands used so far are the only ones required for the simplest of mail merges.

Comparing RV and RU

In order to see the different result obtained by using the 'RU' command, change the 'RV' command to 'RU' in 'EXAMPLE1' and repeat the PRINTS. When using 'RU' the names and addresses will soon get confused and mixed up.

Load the data file back into PROTEXT and go through the records, inserting a blank line at the end of any record that is missing the fourth address line, making each record effectively the same length. Save this amended data file with the name 'DATFILE2'.

Reload the text file 'EXAMPLE1' and change the name of the data file specified by 'DF' to datfile2. Carry out another print, using 'RU' to read the data and this time everything will work correctly, but when replaced with 'RV' things will get confused again.

From the above, it should be seen that the main use of 'RU' is with data files which contain records with the same number of fields in each record. Many database programs construct data files of this sort, which are suitable for use with PROMERGE. Except for the simplest of data files, containing only names and addresses, it is normally preferable to make sure that all records are of the same length, even if it means that either blank lines or lines containing the ' $ ' marker, are used to signify an empty field. Some programs permit you to specify default text, to be used in the event of an empty field being found, in which case ' $ ' would normally be specified.

'RV' can also be used to read data files which contain a constant number of fields, but if any of the fields in a record are null, then 'RV' will interpret these as the end of a record and give a null value to the remainder of the variables, which is not the required result. To see this in action, reload 'DATFILE2' and insert some telephone numbers after the name and before the address. Leave one or two of them as blank lines and save as 'DATFILE3'.

Reload 'EXAMPLE1' and alter the data file name to 'datfile3'. Change the 'RV' line to 'RU' and insert a variable called ' tel '. The line should now look like:-

```
>RU name tel addr1 addr2 addr3 addr4 dummy        ; rem read variables
```

Save this file with the name 'EXAMPLE2' and carry out another PRINTS. Everything will work correctly, as 'RU' reads all the fields, including any empty ones.

Note: When amending earlier examples, to create new ones, do not forget to alter the title in the first comment line to suit the new example number.

Note; It is necessary to read the telephone field and allocate it to a variable, even though it is not actually required for the document. It is essential to read all the fields of a data record, even though only certain of them may be required for the document, otherwise PROMERGE will get out of step with the data records.

Now change the 'RU' to 'RV' and repeat the print. This time things will get out of order again, as 'RV' will cause PROMERGE to stop reading fields as soon as it comes across an empty field.

There are two ways to get round this problem with 'RV' the simplest one being to use the ' $ ' marker in place of an empty line wherever there is an empty field in a record, but this is not always possible with data files created by other programs. The second method requires the splitting of the 'RV' command on to a number of lines and is covered in detail in the chapter on advanced mail merging.

As a general rule, 'RU' is used to read the contents of data files having records which contain a constant number of fields, whatever the format of the file, whilst 'RV' is used for simple data files where the last field in a record may, or may not, be present and for a number of specialised uses, discussed later in the manual.

Creating data by other means

So far, the only way that it has been possible to insert text into a document has been by reading it from a data file, but there are two commands which enable you to construct data in other ways.

>AV (message) var (number)	{(message) var (number) }	Ask variable.
>SV var = string of text	{var = string of text }	Set variable.

NOTE: The usual use of normal ' ( ) ' brackets indicates that the parameter is optional, whilst curly brackets, as before, indicate that optionally, one or more extra items may be specified. ' message ' is a piece of text to be displayed on the screen and ' number ' is a number between 1 and 255.

AV Ask for variable from keyboard.

Examples:
```
>AV "Name", name, 20, "Address line 1", addr1
>AV name, addr1, addr2, addr3, addr4
```

Description: The messages are displayed on the screen as prompts. If no message is specified for any particular variable then the variable name is displayed on the screen followed by a question mark. Text may then be typed in, and this is assigned to the variable. If 'number' is specified this sets the maximum length of text that may be entered. If the number is omitted the maximum length permitted is 255 characters.

SV Set variable to a value.

Examples:
```
>SV firstname="Peter"
>SV name=firstname+"Smith"
>SV counter=1
```

Description: This command automatically sets a variable to a given value, which can then be used in the text in the same way as other variables. The 'string of text' may be either a string of characters, which must be enclosed in quotes, or it may be another variable, in which case the variable name is used, without quotes. It may also be a combination of both, in which case they are joined with a ' + ' sign, as in the second example. Valid decimal numbers may also be used, either with or without quotes.

There are many other things that can be done with the 'SV' command, but they are beyond the scope of this introductory chapter and are discussed in detail in the later chapter on variables.

The easiest way to see the effect of these two commands is to modify 'EXAMPLE2', used earlier in the chapter. The following two lines should be inserted after the line which starts '>RU'.

```
>AV "Enter Date" date
>SV year="1986"
```

and the following line after 'Dear &name& '

```
>CE      Renewal date - &date& &year&
```

Save the modified file as example's and use PRINTS again, to see the results.

The main commands are concerned with simple mail merging and have now been covered but there is one final command which should be mentioned in this chapter, which, whilst not strictly being mail merging in the normal sense, is a simple and yet very powerful command .

IN	Insert file into text.
Example:	>IN file1 >IN file2 >IN file3
Description:	This command may be used at any point in the text of a document and the named file is read from disc and the contents printed. When the end of the file is reached, printing of the original file continues at the line after the IN command. The file is not loaded into memory, merely printed. Any number of 'IN' commands may be issued.
NOTE:	Drive letters may be specified preceding the filename, in the normal fashion.

NOTE: It is not possible to have IN and DF in use at the same time, due to the restriction imposed by the operating system, of only one open input file being permitted at a time .

NOTE: If a document, which is being printed using the 'IN' command, also contains an 'IN' command, an error message will result, as it is only possible to have one input file open at a time. The only exception to this is where the last command in a document is an 'IN' command, in which case the file called by the ' IN ' command Fill be opened and printed.

IN is a very useful command, in that it permits you to print continuously, using a number of files. One of the advantages of this is that page numbering, Headers, Footers, or margins set at the start of the document will follow through correctly, unless altered by any stored commands in the files being printed. In this way, it would be possible to print out a whole book, from beginning to end, in one run, with headings and page numbers running correctly throughout.

Another use of 'IN' is to merge commonly used pieces of text into a document. For example, you could keep a file called ' address ', containing four address, on disc and instead of loading it into memory each time you write a letter, it would only be necessary to put '>IN address' at the start of each letter. As another example, if you frequently have to use the same paragraphs of text in a business letter, these could be saved on disc and called as necessary. Further uses of the 'IN' command are detailed in the chapter on advanced mail merging.

Address labels

Finally, before moving on to more advanced mail merging, this very common and straightforward use of PROMERGE should be mentioned.

```
>CO Address labels
>-------------------------------------------------------------------------------!-------------------------R
>PL 9                    ; Number of lines to repeat label (6 per inch)
>ZM                      ; Set all margins to zero
>DF datfile1             ; Data file to read
>CO Next two lines read contents of two records
>RV name, addr1, addr2, addr3, addr4, dummy
>RV nexname, nexaddr1, nexaddr2, nexaddr3, nexaddr4, nexdummy
!name!                                                  !nexname!
!addr1!                                                 !nexaddr1!
!addr2!                                                 !nexaddr2!
!addr3!                                                 !nexaddr3!
!addr4!                                                 !nexaddr4!

---------------------------------------------------------------------------------------------------------------
```

There are several points that should be noted with the above example.

a). 'PL' should be set to a value equal to the number of lines that are required to move from the first label row to the top of the next label row. The normal setting of a printer is six rows to an inch. In the above example, the setting is therefore for address labels with a one and a half inch repeat.

b). 'ZM' must be used to cancel the default top, bottom, header and footer margins.

c). 'RV' is used, as 'datfile1' contains variable length addresses.

d). An extra blank line is required at the end of the data file, otherwise, if there is an odd number of addresses in the file, the data file will be exhausted before the second 'RV' command is read and the last address will not therefore be printed.

e). If the address labels are positioned in rows of three, another blank line will be needed in the data file and a further 'RV' command required to read the data.

f). Note also that the number of lines which will be printed, including blank lines at the end of the text, must be the same or less than the number specified as the page length.

7. CONDITIONAL PRINTING AND MAIL MERGING

Conditional printing and conditional mail merging are really two different things, but, as they are frequently used together, both will be covered at the same time.

A conditional mall merge is one where text will be either printed or not printed, depending on whether certain conditions are met. It may affect whether all or only part of the text is printed, or even whether one piece or another will be printed instead.

A little thought will give some idea as to the enormous power of these commands and once the principles are understood, some time spent experimenting will soon bring to light endless possibilities. There are certainly far too many 'to cover them all in this manual.

There are four commands which are directly concerned with straightforward conditional printing, tome of which can be used on their own, but most of which are used in conjunction with each other.

>IF condition	If condition is true then print block.
>EL	Else print other block if IF condition false.
>EI	End of an IF block.
>SK condition	Skip copy if condition true.

Before considering these commands, it is important to have a thorough understanding of what is meant by 'condition'. A condition is where a string (piece of text), a number, or a variable, is compared with another string, number, or variable. The result will be either 'true' or ' false'. For the purpose of these commands, there are eight possible comparisons that can be made:-

=	equal to
<>	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
IN	is contained in (e.g. "jo" in "Mr Jones" is true)
NOTIN	is not contained in

Decimal numbers are compared numerically. A numeric argument is one which the program recognises as a valid decimal number, such as '37' or '5.4'. If a valid number is found, it will be compared numerically, otherwise it will be compared character for character. 'IN' and 'NOTIN' will automatically carry out a character comparison, but the other comparators will attempt a numeric comparison first, before defaulting to a character comparison if the comparison is numerically invalid. For the purposes of a character comparison, no account is taken of whether a letter is upper or lower case and both are considered equal.

The first and simplest of these commands is:

IF Print if condition true.

Example: >IF "Married" NOTIN status
>IF firstname IN fullname
>IF age > 40 (eg. true if age is over 40)

Description: The condition is tested and if true the block of text which follows is printed (and any stored commands acted upon). The block of text extends until either an 'EL' command, an 'EI' command, or the end of the text. If the condition is false then everything within the block is ignored, including stored commands.

Note: IF blocks may be nested to a depth of 7.

EI Endif. End of IF block.

Description: This command marks the end of an 'IF' block. If the 'IF' block was nested within another, then printing reverts to the remainder of the previous 'IF' block, otherwise unconditional printing resumes.

Note: Every 'IF' command must have a matching 'EI' command.

Load 'EXAMPLE3' and add the following lines to the end of the text, save as 'EXAMPLE4' and again use PR INTS to see the effect:-

```
>IF tel > " "
        If you would be kind enough to telephone us . . . .
>EI
```

You should find that any of your records which contain a telephone number will have the extra line of text added to them, but if the telephone field is empty, then it will be omitted.

EL Else.

Description: The block of text following this command is printed if the condition in the previous IF command was false, but is ignored if that condition was true. The block extends to an EI command or the end of the text.

Note: IF blocks may be nested within EL blocks.

'EL' is a very useful command, as it permits the inclusion of an alternative piece of text, to be used in the event that an 'IF' condition is found to be false. "IF this condition is true then print this block, ELSE print the next block".

Add the following lines to 'EXAMPLE4 immediately before the '>EL' command, save as 'EXAMPLE5' and again use PRINTS.

>EL
If you would be kind enough to write to us . . . .

You should find that this time either one or the other message is printed, depending on whether a telephone number is present in the field or not.

SK Skip printing if condition true.

Examples:
>SK "Mr" IN name - If name contains 'Mr', not printed.
>SK add4="London" - If add 4 is "London" not printed.

Description: The condition is checked and if true the remainder of the text is skipped. This allows the selective use of data from a file.

Note: The same effect could be achieved by the use of 'IF' but 'SK' is preferable, where it is required to omit printing the remainder of the text for that record, as the effect is immediate, whereas 'IF' will continue to check through the document for further commands, until the end of the document is reached.

The easiest way to see how this command works is to insert the following line into 'EXAMPLE5' immediately after the '> RU' line and save it as example's before printing.

> SK addr4 > " "

This line tests to see whether the variable 'addr4' contains anything or not and if it does, then the condition is true and the remainder of the document will be skipped. In this case, as the test is made before anything is printed. the complete record will be skipped. The effect of this is that only records with three address lines will be printed. Another example would have been to use ' >SK tel=" " ' which would have resulted in only records containing a telephone number being printed.

The commands covering the simplest forms of conditional mail merging have now been covered, but there are several other commands which extend the possibilities further and these are covered in the chapter on advanced mail merging.

To make the most of PROMERGE, it is recommended that some time is spent experimenting with the various commands covered so far, before continuing with the chapter on advanced mail merging.

8. ADVANCED MAIL MERGING

Many new commands were introduced in the earlier chapters and in this chapter a number of now commands will be introduced, as well as extensions to commands already covered in their simpler form.

New commands

>CF	Close data file.
>ID var	IF variable defined then print block.
>IU var	IF variable undefined then print block.
>RP	Repeat.
>UN condition	Until the condition is true.

In earlier chapters the ' DF ' command was used to open data files for reading. One of the restrictions mentioned was the fact that the operating system will only permit one file to be open for reading at a time. For many purposes this will be no problem, but there can be a number of situations where it would be useful to open another data file, having extracted the required information from an earlier file, or perhaps to make use of an ' IN ' command to insert a standard piece of text. ' CF ' is provided expressly for the purpose of closing a data file, enabling another to be opened in following commands.

CF	Close data file.
Example:	>CF
Description:	CF closes a previously opened data file, permitting a further file to be opened for reading.
Note:	A CF command must be issued between two DF commands, otherwise the second DF command will be ignored.

The use of the ' IF ' stored command has already been explained in the chapter on conditional printing, but in addition to ' IF ' there are two further commands which can be used instead of ' IF ' for specialised uses. These are:

ID	If defined.
Example:	>ID var1
Description:	ID is a special form of the IF command, which returns ' true ' if the variable has been defined, and ' false ' if not. If true, the text following will be printed, or any stored commands acted upon, until the matching ' EI ' is found. This command is very useful when used in conjunction with AV.

IU IF undefined.
Example: >IU variable

Description: IU is the opposite of ID, in that if the variable has NOT been defined, then the condition is true.

A variable has been defined if it has already been given a value earlier in the text printing, by any of the commands, AV, RU, RV, SV. This allows, for example, an AV command to be executed only once and the resultant value to be used for the remainder of the mail merging.

Reload 'EXAMPLE4' and make the following alterations:

Insert the line - ' >IU date before 'AV "Enter date" date'
and add - ' >EI ' on the line following ' >SV year="1986" '.

Save the text as 'EXAMPLE7' and using PS it will now be found that at the start of printing, the date will be requested. This has the effect of defining the variable ' date ', so that in subsequent copies, as ' date ' has now been defined, the date will not be requested, the existing value being used for all further copies.

' ID ' works in exactly the same way, but in reverse, action only being taken if the . specified variable is defined.

PROMERGE also provides the means to carry out repetitive actions in a document. These are carried out using the following commands.

RP Repeat.
Example: >RP

Description: ' RP ' marks the start of a block of text that will be repeated until a condition specified in ' UN ' is found to be true. Normal printing will then continue.

UN Until a condition is true.
Example: >UN "N" < name

Description: ' UN ' marks the end of a Repeat - Until loop.
When PROMERGE finds this command, it checks to see if the specified condition is true and if not, jumps back to the earlier ' RP ' command. This sequence will repeat until the condition is true, at which point the loop will be exited and normal printing of the rest of the document will continue.

Note: A Repeat - Until loop will always be carried out at least once.

Note: When RP UN is used to carry out printing, it should be remembered that any natural page breaks may be altered and it may often be desirable to use a forced page break (' >PA n ') at some stage after completion of the loop. .

These commands can be used in a number of ways, either to repeat a block a set number of times, or until a certain condition is met.

Example to create a data file:

```
>ZM
>RP
>AV "Enter name " name
>IF name <> " "
>AV addr1 addr2 addr3 addr4
>SV dummy=" "
! name !
! addr1 !
! addr2 !
! addr3 !
! addr4 !
! dummy !
>EI
>UN name=" "
>ST
```

In the above example, the name is requested and checked to see that it contains text. If so, then the address is requested and the text printed, before looping back to repeat the process. If ' name ' contains nothing, then the check in ' UN ' will cause the loop to be left and printing to stop. Using PRINTF will create a file suitable for use as a data file.

Example to request information a given number of times.

```
Description of goods        Unit cost          Quantity          Discount

>SV counter=0
>RP
>AV descrip cost number disc
>SV counter = counter + 1
>----------------------------------------.----------------------.----------------------.------------R
&descrip&                  &cost&         &number&                    &disc&
>UN counter = 10
```

In this example a variable called 'counter' is set to a value of zero and when the text is printed, requests are made for a description of the goods, the quantity and the discount. The value of the variable is increased by one and the loop repeated until the value of 'counter' reaches ten, when the lopp will be left and the remainder of the text printed.

More on the RV stored command.

In the earlier chapter on simple mail merging, many of the differences between the 'way that ' RU ' and ' RV ' function were described, as well as the fact that ' RV ' was particularly suitable for use with the simplest of data files. Mention was also made of the fact that It could be used to read more complicated data files and in fact had definite advantages for certain purposes.

It will be recalled that one of the special features of ' RV ' was the fact that as soon as it came across an empty field (signified by two consecutive CRs), it ceased to read any further data fields and allocated a null value to any further variables on that command line. This was also one of the things that caused problems when trying to read records which contained empty fields, as it automatically gave a null value to the remaining variables, even though they were present in the record. This also caused the reading of the records to get out of step.

One solution was to make use of the special PROMERGE indicator ' $ ' to indicate to PROMERGE that the field was present, but with a null value. In this way ' RV ' would continue to read the remainder of the variables.

There is another solution to the problem, which can be used where the records contain empty fields, even without the ' $ ' marker, as would be created by some other programs. The answer is to split the reading of the variables into a number of lines, in such a way that any field which may be empty is the last field on that command line. Several lines can be used if there is a possibility that a number of fields might be empty.

In the following example it is assumed that the data file contains records in which the second, sixth and seventh fields might be empty in some records and that each record consists of eight fields plus a blank line.

```
>DF datafile
>RV name tel
>RV addr1 addr2 addr3 adrr4
>RV occupation
>RV salary dummy
```

When RV reads ' tel ', if it finds it to be empty it will give it a null value, but as there are no more variables on that command line, PROMERGE will continue to the next line and continue to read the next few variables. As soon as a null value is discovered, the remainder of the line will again be given null values and so on to the end.

More on the IN stored command.

In the chapter on simple mail merging a description was given of the use of the 'IN' command to permit the consecutive printing of a number of files, whilst still maintaining continuity of page numbers, headers, footers etc. Mention was also made of the fact that ' IN ' could be used to cause files to be printed at any stage in the printing of a document, permitting the merging of commonly used blocks of text for example.

A third use briefly mentioned was the ability to use ' IN ' at the end of a document to call another document. This can be repeated as many times as required. In general, a more satisfactory way of doing it would be to use the previously mentioned list of '>IN filename' commands, however.

One further use of ' IN ' which can be very useful is to use it in a recursive fashion, to call itself . This will permit the same document to be repeated as many times as required. The method of doing this is very straightforward and simply involves using the '>IN filename' command at the end of the file and then saving the file with the name used in the ' IN ' command. In this way, when the end of the file is reached, it automatically calls itself .

Displaying messages during printing

So far, there has been no means of knowing which record is next to be printed, until it is actually being printed. There are some occasions when it would be convenient to decide at printing time whether a record was to be printed, or even what was to be printed. PROMERGE provides a number of commands to display messages on the screen. These commands are ' CS ', ' DM ', ' ST ' and ' WT ' which have already been described in the chapter on stored commands. What was not mentioned was the fact that the variables could be used in messages, thus providing a method of displaying their contents.

Examples:
```
>DM   The next record is &name&
>CS   Do you want to print the record for &name&?
>WT   Print &name&? ESC to stop or any key to print.
>ST   There are no more records to print called &name&.
```

In all cases the variable must have been defined before the message line and following the message line or lines, suitable stored commands should be inserted to take the appropriate actions.

Example: Modify 'EXAMPLE2' to the form shown below and save as 'EXAMPLE8'

```
>CO EXAMPLE8
>PL 24
>DF datfile3
>RU name tel addr1 addr2 addr3 addr4 dummy
>CS    Do you want to print the record for &name&?
>AV " "print
>SK print <>"Y"
&name&
etc

>DM    Copy to &name& has now been printed
>WT    Press any key to continue
```

The above example goes through the complete data file, prompting with the name of each record and asking whether it is to be printed or not and displays messages after printing, to inform you who it was for and to ask you to ' Press any key to continue '.

9. MAKING THE MOST OF VARIABLES

There are a number of things that can be done with variables, which have not yet been covered and this chapter introduces methods of manipulating the contents of variables in various ways, as well as introducing a number of other uses for variables.

Joining variables and text

When ' SV ' was introduced, mention was briefly made of the fact that strings of text or variables could be joined together to make one variable. This can be particularly useful on many occasions. In 'EXAMPLE3' ' SV ' was used simply to give a variable a value and then two variables were positioned next to each other in the text, so that when printed, they appeared as one.
e.g. &date& &year&. It could have been done more simply by joining the two at an earlier stage and then using just the one variable in the text.

For example:

```
>SV     year="1986"
>AV     "Enter Renewal Date " date
>SV     date=date+year
```

or more simply:

```
>AV     "Enter Renewal Date " date
>SV     date=date+" 1986"
```

There are numerous uses to which this ability to join variables and strings of text can be put, but one in particular is worth mentioning.

No punctuation was used in the addresses of the earlier examples, with the exception of a full stop at the end of the first sentence, for a very good reason. Frequently the effect of "personalised" circulars is spoilt by unsightly gaps in the text and misplaced punctuation marks. PROMERGE takes care of the gaps automatically, but selecting the correct punctuation is a bit more difficult.

Commas and full stops could have been placed in the text, after each of the variables, so that they were printed. For many simple mail merges this would have worked satisfactorily as long as all the variates contained text. If some of the variables contained nothing, the result would be punctuation marks all on their own, or perhaps, two together. When printing 'EXAMPLE2', it may have been noticed that sometimes the full stop at the end of the first sentence was next to the address and sometimes it was separated because the last address line was missing.

Another way would be to put the punctuation in the data file as part of the text for each field, with the fields wrapped in quotes. In many cases this would work satisfactorily, but it has a number of disadvantages, not only because it becomes necessary to enclose fields in quotes in the data file, but because the situation can arise where the punctuation mark is not the one required in the text.

The solution to this is simple, once the principle is understood. Using 'EXAMPLE2' from the earlier chapter, if the record was one where the address consisted of only three lines, then the third (and last) line of the address would require a full stop, but if the address consisted of four lines then the third line would require a comma and the fourth line a full stop. The easiest way to understand this is to look at an example of what is required. Load 'EXAMPLE2' and modify it before saving the modified version as 'EXAMPLE9' :-

```
>CO EXAMPLE9
>PL 24
>CP OFF
>DF datfile3
>RU name tel addr1 addr2 addr3 addr4 dummy
>IF addr4 > " "
>SV addr3=addr3+"," addr4=addr4+"."
>EL
>SV addr3=addr3+ "."
>EI

&name&,
&addr1&,
&addr2&,
&addr3&,
&addr4&

Dear &name&,
        Thank you for your letter about the insurance for &addr1&, &addr2&, &addr3& &addr4& We think that you will find the rates quoted . . . .
```

In this example, we know that 'name' 'addr1' and 'addr2' will always contain something, so it is sufficient to simply insert the punctuation in the text for these variables, but the ' IF ' command checks to see whether ' address4' is an empty variable. If not, it adds a comma to 'addr3' and a full stop to 'addr4' otherwise ' EL ' puts a full stop on the end of ' addr3 ' instead. In this way, when printing takes place, the appropriate punctuation is printed as part of the variable. If there had been a possibility that an address could have consisted of only two lines then an ' IF ' command would have checked to see whether 'addr3' had been a null string, before checking added .

Splitting variables

As well as joining variables and text together, PROMERGE provides the facility to split the contents of variables down into smaller parts, either by character or by word. This is achieved by specifying which characters or words are required in the following way:

var[a : b]	from character 'a' to character 'b' inclusive.
var[a :]	from character 'a' to the end.
var[a]	character 'a' only.
var[Wa : b]	from word 'a' to word 'b' inclusive.
var[Wa :]	from word 'a' to the end.
var[Wa]	word 'a' only.
var[W-1]	the last word only.

If only numbers are specified, the PROMERGE will split the contents on a character basis, but if the first number is prefixed with a letter ' W ' then the variable will be split by words. The last option is a special case and wiIl select the last word only.

Examples:

If a variable called ' name ' contains 'Robert Stephen Smith', then the following will be true : -

name [3 : 6]	is	' bert'
name [10 :]	is	'ephen Smith'
name [3]	is	'b'
name [W1 : 2]	is	' Robert Stephen'
name [W2 :]	is	'Stephen Smith'
name [W2]	is	'Stephen'
name [W-1]	is	'Smith'

Using the above variable as an example, the following values for the new variable will result:-

>SV newname=name [W1] + " " + name [W-1]	gives 'Robert Smith'
>SV newname=name [W2 :]	gives 'Stephen Smith'
>SV newname=name[1 : 3] + " " + name [W - 1]	gives 'Rob Smith'
>SV forenames=name[W1 : 2]	gives 'Robert Stephen'
>SV newname="Mr " + name [W - 1]	gives 'Mr Smith'

It is also possible to use this form of splitting when checking conditions, using ' IF ' ' SK ' and ' UN '.

Examples:

>IF "Smith" IN name[W - 1]	; block is printed.
>SK name[W - 1] > "M"	; remainder is not printed
>UN name[W1] = "Robert"	; repeat loop ends.

Using variables in stored commands

The use of variables in the stored commands concerned with displaying messages has been covered in the previous chapter, but it is also possible to use them in the majority of other stored commands, but in a much more sophisticated fashion.

The contents of a variable can be used in a stored command to define conditions, values and even other variable names in other stored commands. The easiest way to describe the use of this facility is by listing a few of the possibilities, but a full description of all the options is beyond the scope of this manual and it is suggested that some time spent experimenting with stored commands and variables will bring to light numerous possible uses.

Examples:

```
>AV  "Name of file " filename
>IN  &filename&

>AV  "Start at page " startat  "End at page " endat
>SA  &startat&
>EA  &endat&

>AV  "No. of copies " copyno
>NC  &copyno&

>AV  "name to find " find Operator "Data field " field
>SK  find &operator& field

>AV  "Continuous printing Y/N " cprint
>IF  "Y" in cprint
>SV  cprint="ON"
>EL
>SV  cprint="OFF"
>EI
>CP  &cprint&
```

Numeric arithmetic

Numeric arithmetic is also possible with PROMERGE and calculations can be carried out using the ' + ', ' - ', ' * ' and ' / ' operators. Compound calculations are permitted and the calculations are carried out in the order that they are encountered rather than by operator priority. Brackets may NOT be used to force calculations. Invalid numeric contents of variables will result in an error message and printing halting.

Calculations can be carried out on the contents of variables, as long as they are of numeric type and the results stored in further variables.

All calculations are accurate to two decimal places and the results are substituted into the text showing two decimal places. The only time a number will be displayed without the two decimal places is if the number was not the result of a calculation. For example, if ' cost ' is defined by ' >SV cost=9 ' then ' &cost& ' will display ' 9 ' . To display ' 9.00 ' add the command ' >SV cost=cost+0'.

Example :

```
>AV      price quantity discount
>IF      discount=" "
>SV      discount=0
>EI
>SV      disccalc=100-discount/100
>SV      total=price*quantity*disccalc
>SV      vatcalc=1.15
>SV      totincvat=total*vatcalc
>SV      pound=totincvat [ W1 ]
>SV      pence=totincvat [ W-1 ]
```

The above example illustrates a number of possible ways to manipulate and calculate numeric variables, including an example of obtaining the equivalent of INT (num) and FRAC (num).

Numeric arguments can also be used for making comparisons and in this case ' > ', ' < ', ' <> ', ' >= ', ' <= ', and ' = ' can all be used.

Example:

```
>IF age > 40        - true if age is greater than 40
```

Variables can also be used as counters and markers in stored commands, to cause actions to happen after a certain number of times.

Example:

```
>SV      counter=0
>RP
>AV      description cost quantity
>SV      total=cost*quantity   counter=counter+1
>----------.---------.----------.----------.---------R
&description& &cost& &quantity& &total&
>UN      counter=10
>CO remainder of text follows
```

In the above example, the counter is set to zero at the beginning and a description, details of cost and the quantity are requested. ' SV ' then calculates the total cost, assigns it to a variable called ' total ' and increments the value of ' counter '. The next line prints the variables and ' UN ' checks to see whether the value of ' counter ' has reached ten. If not, then the loop is repeated, otherwise the loop will be exited and the remainder of the text printed.

10. PRINTER COMMANDS

Printer control commands

This group of commands is principally concerned with the sending of control codes to the printer, to change the various settings of the printer. The commands ' CC ' and ' RC ' are included mainly for completeness, but the best way to alter control codes and character definitions remains SETPRINT and the use of a printer driver.

>CC a {n} (; {n})	Define printer control code.
>OC n {n}	Output codes to printer.
>PR filename	Load printer driver of that name.
>RC a {n}	Redefine character.

CC Redefine control code.

Examples:
```
>CC "h", 27, 82, 6 ; 27, 82, 3
>CC @
```
The second example stops the reset code being printed. This should be used on a printer that is not Epson-compatible and has no reset code (or you don't know what the reset code is).

Description: Any printer control code (@, a-z) may be defined as in SETPRINT. The first item after ' CC ' is the control code letter, followed by the sequence of codes for ' on ', followed (optionally) by a semicolon and then the codes for ' off '.

OC Output codes.

Examples:
```
>OC 27, " 3 ", 12 ; change line spacing
>OC 27, J, 20
```

Description : This command takes a list of codes which are directly sent to the printer. It is particularly useful for defining character matrices on the printer when a long string of codes are required. The printer driver could be used, but can only hold a limited number of characters.
The codes may be sent as decimal numbers, hex numbers (prefix with & or #) or ASCII characters. If an item is not a number it is sent as an ASCII character. To send the ASCII code of a decimal digit enclose the digit in quotes. Codes may be separated by commas or spaces.

IMPORTANT NOTE: The Amstrad computers have a 7 bit parallel printer interface. It is IMPOSSIBLE to send numbers above 127 to the printer. The printer probably has a code that will cause all following numbers to have 128 added to them, but this is of no use for defining characters.

PR Load printer driver.
Example: >PR taxan
>PR B:letter

Description: The named printer driver is loaded. AII the control code and character definitions are set to the new values, as are all the print options (margin sides, etc.), Drive letter prefixes are permitted.

RC Redefine character.
Examples: >RC £ 27 ">" 1 27 "="
>RC "!"="!" 8 "."

This is equivalent to using the 'redefine character ' option in SETPRINT. The first item after ' RC ' is the character to be redefined, this is followed by the sequence of codes to be printed whenever the character occurs. The codes may be entered in the same way as for the ' OC ' command, above.

Microspacing

When PROTEXT right-justifies a line, the extra spaces are spread out along the line as evenly as possible. However the spaces always occur in whole characters. Microspacing is a means of dividing the spaces evenly between the words on a line, and so gives a better appearance to the text. There is a cost - printing is much slower than when microspacing is being used.

>MS ON/OFF	Microspacing on/off.
>MC n { , n }	Define microspace code sequence.
>CW n	Define microspace character width.

On Epson-compatible printers, using microspacing is very easy. Put the stored command ' >MS ON ' at the start of the text and microspacing will be enabled. ' >MS OFF' will turn it off , so it is possible to print a document using microspacing on just a part of the text.

Note: If right justification has been disabled (with >RJ OFF), then microspacing will not work.

The above will work only with normal size print (pica) . If you are using different size characters the CW command must be used to define the width of the microspace character. This is done with a number that is the width of the character in multiples of 1/120th of an inch. The values that should be set for the different print sizes are as follows:

Pica	12 (this is the default setting)
Elite	10
Condensed	7

If the text is being printed enlarged, then these values should be doubled. (These figures are given for Epson printers) .

On other printers, PROTEXT must be given a code sequence to print a microspace. This is the sequence of codes that moves the printer head by the smallest possible amount. The way this is done will be different for different printers (and it may not always be possible) . As an example the following definition could be used for Epson printers (though it is not necessary since microspacing already works on Epsons) :

```
>MC  27  "L"  1  0  0
>CW  12
```

The codes 27, " L " put the printer into 'double density graphics bit image mode'. The next two bytes are the number of bytes of data to be printed, low byte first (i.e. 1 byte of data) . The last zero is the one byte of data. The effect is that the print head is moved 1/120th of an inch to the right.

The definition of the character width setting, as being in multiples of 1/120th of an inch, assumes that it is possible to move the print head by 1/120th of an inch. If the smallest possible print head movement is different from this, and the microspace code has been defined to move the printer by this amount then the definition of character width is 'The number of microspace codes that need to be printed to move the printer by the width of a character'.

Printing in different languages

>DA	Switch printer to Danish.
>EN	Switch printer to English.
>FR	Switch printer to French.
>GE	Switch printer to German.
>IT	Switch printer to Italian.
>SP	Switch printer to Spanish.
>SW	Switch printer to Swedish.

To simplify printing text in mixed languages, stored commands have been added to select the various character sets on the printer. These commands have several effects: the printer is switched to the appropriate character font, the font-change control codes (f and g) are changed to the definition appropriate to the chosen language, and the # and £ character definitions are changed. The definitions of these codes and characters are the same as they would be if the language command had been used. The appropriate PROTEXT language command should be used to display the various character sets on the screen. The main advantage of using these stored commands is that the text will be printed correctly, irrespective of which language is currently being displayed on the screen .

These commands only work on Epson-compatible printers. On other printers the same effect can be achieved by using >OC to select the character font, and >RC and >CC to redefine the control codes and # and £ characters (if these are being used).

A full description of the control code definitions for each language will be found in the technical appendix.

Examples and applications:

1 . Printing in one language only, e.g. German users.

The >GE command is not required. Set the DIP switch on the printer so that the German character set is permanently in use. Enter the PROTEXT command ' GERMAN ' (or ' DEU ') so that German characters are displayed on the screen. It may be found convenient to re-define some of the keys - see the section on configuration of PROTEXT.

2. Printing a document in a different language from normal use, e.g. an English user writing a letter in French.

Put >FR at the start of the document. Enter the PROTEXT command ' FRENCH ' to set the screen to French. The stored command ' FR ' ensures that the French characters will always be printed, even if the document is printed whilst in English mode. This is not the case if the font change control code ' f ' is used.

3. Printing a mixed language document, e.g. an English user writing a document that is part-English, part-French, part-German.

(a) If the different languages are contained on different lines, the appropriate stored command should be issued before the section in a new language, e.g.

```
> FR
. . . . text in French . . . .
>EN
. . . . text in English . . . .
>GE
. . . . text in German . . . .
```

If languages are mixed on a single line, then the above method can still be used. At a point where the languages are to be changed, the following should be done:

```
>LS 0
. . . . text in English . . . .
>FR
>LS 1
. . . . text in French . . . .
```

(b) If there are many language changes, this method may be unwieldy. In this case the font change control codes should be used, but bearing in mind that these are defined differently, depending on the language. To ensure that the document is always printed correctly, the font change codes can be defined explicitly within the text, e.g.

```
>CC " f " 27, 82, 1 ; 27, 82, 3
>CC " h " 27, 82, 6 ; 27, 82, 3
```

With this example, CTRL-X F should be typed at the start of a French section, and CTRL-X H at the start of an Italian section. In each case the second occurrence of the control code in the text will cause printing to revert to English. The codes used in this example are suitable for use with Epson compatible printers and may require changing for other printers.

11. COMMAND MODE

PROMERGE provides a number of completely new commands and also several extensions to existing commands.

1. EXTENDED PROTEXT COMMANDS

HELP (H)

The PROTEXT HELP command is extended to list the additional commands, on a separate page of information, accessed from the main help page.

PRINT (P)

Syntax: P (infile)

Description: This command now allows printing directly from a file if a filename is specified after the command name. The file will be printed directly from disc. It will not be loaded into PROTEXT and will not corrupt any text in memory. If no filename is specified , the file in memory will be printed, as previously in PROTEXT.

One possible use for this command is when used in conjunction with a printer buffer, in which case the buffer can be loaded with the file using this option, after which it is possible to continue editing the text in memory, whilst printing of the other file continues.

Note: PROMERGE PLUS users will find full details about a slight change in the use of ' P ' and about printer buffers and background printing in the next chapter.

PRINTF (PF)

Syntax: PF (outfile) (infile)

Description: With PROMERGE active, PF creates a pure ASCII file with no control codes at all. Tabs are expanded to spaces and page breaks and margins are still shown. If these are not required, the stored command ' >ZM ' should be inserted at the start of the file, to set all margins to zero.

Note: If only one filename is specified, a file with the specified name will be created from the document in memory. If no filename is specified, then both an input and an output filename are prompted for.

Note: Filenames may be prefixed with the drive letter in the normal fashion. eg. ' PF B:outfile A:infile' will create an Ascii file called 'outfile ' on drive B from a file on drive A called ' infile '.

Printing from one file to another is therefore a means of converting a file into an ASCII file without loading it into memory.

PRINTS (PS)

Syntax: PS (infile)

Description: This command now allows printing to the screen directly from a file, if a filename is specified after the command name. The file will be printed directly from disc. It will not be loaded into PROTEXT and will not corrupt any text in memory. If no filename is specified, the file in memory will be printed, as previously in PROTEXT.

Note: If ESC is pressed during a PRINTS of a file in memory, on returning to edit mode the cursor will be positioned at the point that had been reached in the text. This makes it easy to correct mistakes noticed whilst printing to the screen.

This feature is very useful for reviewing the contents of a file on disc, without the need to load it into memory.

VERSION (VER)

Description: The version numbers of PROTEXT and PROMERGE are both listed.
This is the easiest way to check that PROMERGE is active.

2. NEW COMMANDS

PRINTQ (PQ)

Syntax: PQ (infile)

Description: When ' PQ ' is used, the effect is to cause printing to take place in ' Near Letter Quality ' (NLQ), without the need to alter SETPRINT, or to insert the NLQ control code into the text. Otherwise it is exactly the same as PRINT. If the optional file name is stated, the file will be printed from disc, otherwise the text in memory will be printed.

Note: This command will only work with printers which support NLQ mode and the control codes in SETPRINT must be set to the correct sequence of the printer codes. The default values suit most Epson compatible printers.

PRINTQB (PQB)

Syntax: PQB

Description: Serves the same purpose as ' PRINTQ ', but only on a block of text selected by use of the block markers.

Note: It is not possible to use this command when printing from a file on disc.

SPLIT

Syntax : SPLIT filename (number)

Description: This command will split a text file that is too large to fit into PROTEXT into a number of smaller files. The name of the file to be split must be specified. Optionally, the maximum size (in bytes) required for the new files may be stated. The default is 8K (8192 bytes). The new files will have the same name as the original file, but with an incremented number in the filename extension.
e.g. 'FILENAME.1' 'FILENAME.2'. The new files will be saved onto the currently selected disc drive, which means that a large file can be split from one disc onto another by specifying the drive number in the command. E.g. ' SPLIT B: filename' will split the file on drive B into smaller files on drive A.

Note: PROMERGE will always attempt to split the file at the end of a line, rather than at exactly the number of bytes specified .

Note: Files may still be printed as one file, by making use of the ' >IN filename' command, described in earlier chapters, to link them together. Files may be joined together to create one large Ascii file, by using the ' PRINTF ' command, in conjunction with ' >IN'.

There are a number of situations where this command is useful, for example, when a text file created with PROTEXT will no longer fit in memory because either PROMERGE or PROSPELL are installed, or with files created by some other programs.

TYPEWRITER (TW)

Syntax: TW (number)

Description: This command puts the computer into direct typing mode. Everything that is typed is printed as soon as the RETURN key is pressed. If a number is specified with the command then that number of spaces is printed before the text. Press ESC to return to PROTEXT. If Near Letter Quality or any font other than normal draft mode is required, the printer must be set up correctly before entering TW. If a document has just been printed in NLQ, then TW will continue in NLQ. Alternatively, many printers have methods of easily switching into NLQ.

Note: This command defaults to zero margins, meaning that if text is normally printed with a side margin of 5, then it will be necessary to specify 5 when entering TW, to maintain the same left margin.

Note: PROMERGE PLUS users will find it is not possible to use this command whilst background printing is in operation, for obvious reasons!

There are a number of possible uses for this command, including the easy addressing of envelopes, or adding postscripts to the end of letters. TW can also be used In conjunction with the extended use of '|P ', (See item 4 below), enabling easy use of the printer, without the need to actually enter PROTEXT.

OPTION (OPT)

Syntax: OPT nn

Values of OPT:

OPT 0	Reset to defaults	(no bits set)
OPT 1	Box mode on	(set bit 0)
OPT 2	Help on	(set bit 1)
OPT 4	Page mode off	(set bit 7)
OPT 8	Ruler line off	(set bit 3)
OPT 16	Right justify off	(set bit 4)
OPT 32	Wordwrap off	(set bit 5)
OPT 64	Overwrite mode on	(set bit 6)
OPT 128	Program mode	(set bit 7)
OPT 256	CTRL-T (Tab/CR on)	(set bit 8)
OPT 512	CTRL-V on	(set bit 9)
OPT 1024	CTRL-S (Spaces on)	(set bit 10)

Description: OPTION will set the state of any of 11 different PROTEXT features. The number in the command determines the feature t is being set. Setting a bit causes a feature to be changed m the default setting. Any bits that are not set will cause the corresponding feature to be restored to the default. Thus ' OPT 0 ' will restore all settings to their defaults. The main use of OPTION is in conjunction with |PROTEXT - see below.

Note: PROMERGE PLUS users should note that OPT sets the features for both files in memory.

3. EXTERNAL COMMANDS

External commands are commands that can be called from another program, usually Basic and are prefixed with a bar symbol (|).

PRINTER Loads a printer driver.

PROTEXT (P) To enter PROTEXT, also extended, see ' Configuration of PROTEXT '.

ROMON7 Turns off all ROMs except the AMSDOS and Basic ROMs.

4. CONFIGURATION OF PROTEXT

A major addition to PROTEXT is the facility to execute any PROTEXT command without entering PROTEXT. Thus a BASIC program can call any commands available in PROTEXT command mode. This can be used to configure PROTEXT automatically.

To execute a PROTEXT command, enter the command as a string parameter in the |PROTEXT (or |P) command. The command will be executed but PROTEXT will not be entered. Any number of commands can be called in this way. A simple |P command will be required finally to enter PROTEXT.

Note: PROMERGE users may insert these configuration commands into the disc file called ' DISC2.BAS ' which is provided for this purpose. Users of PROMERGE PLUS can incorporate them into a Basic loader program, saved with the name ' DISC' , which may then be used to configure PROTEXT.

Example to set up PROTEXT, load a template file and enter PROTEXT:

```
10      |P, "OPT 80" : REM set overwrite mode, right justify off
20      |P, "GERMAN" : REM select German characters for the screen
30      |P, "L LETTER" : REM load a temples: file called ' LETTER '
40      |P : REM enter PROTEXT
```

Example of a Basic program to count the words in a series of files:

```
10 WHILE -1
20 INPUT "Enter filename " ; name$
30 PRINT name$ ; " " ;
40 |P, "L  " + name$
50 |P, "COUNT "
60 PRINT
70 WEND
```

Note: The example to count words is not suitable for use with Basic V1.00, as used by the CPC464.

12. PROMERGE PLUS

PROMERGE PLUS contains all the features of PROMERGE that have already been covered in earlier chapters, but also contains a number of other major extensions to PROTEXT and PROMERGE.

1. Extended command mode entry
2. Two file editing
3. Box operations
4. New and altered printing commands
5. Background printing
6. Calculator
7. File conversion
8. Miscellaneous

1. Extended Command entry

The input routine in command mode has been extended to allow the use of copy cursor editing facilities, as in BASIC. Using the cursor keys together with SHIFT will move the copy cursor around the screen. Pressing COPY will copy the character under the copy cursor to the current input line.

The command line may also be edited by moving the cursor left and right, and inserting and deleting characters. CLR will delete the character under the cursor, DEL will backspace and delete, CTRL-TAB will switch between insert and overwrite modes in a similar fashion to BASIC. CTRL-E will delete all characters from the cursor position to the end of the line (just as in edit mode) .

Pressing the COPY key when the cursor is at the start of a command line now has the effect of recalling the last command used (as long as it was four or more characters long) and repositioning the cursor at the end of the command. This has a number of uses, such as carrying out multiple saves of the same file, or trying to load a file for a second time, if it was not found first time, either because the wrong drive was selected, or the wrong disc was inserted. The reason for the need for the command to be four character: long is to enable short commands to be used, yet still recall the previous command.

e.g.	L file	(not found - wrong drive)
	B	(change drive)
	CAT	(catalogue files)
	< COPY >	(repeat load command)

A number of external commands that have the same name as PROTEXT commands may now be used without prefixing the command with a " | ". In particular this allows the following commands to be used if UTOPIA is fitted : CAT, FORMAT HELP, LOAD and SAVE. If one of these commands is entered with no parameters, or with the correct number of parameters for the PROTEXT command, then the PROTEXT command is executed, otherwise the external command is executed.

Examples:	CAT B	catalogue drive B
	HELP n	list commands for ROM number n
	FORMAT B	format disc in drive B

2. Two File Editing

PROMERGE PLUS now permits two documents to be present in memory at the same time. These documents are maintained quite separately and are loaded and saved individually. Any operation can be carried out on one document without affecting the other, the cursor location and all markers being maintained for each document. Blocks of text can be transferred between one document and the other.

This is an extremely powerful function and is controlled by only three commands, one of which is used from command mode and the other two from edit mode.

SWAP (SW)	: Command mode	- Swap between two documents in memory
CTRL-U	: Edit mode	- Copy block from the other document
CTRL-Y	: Edit mode	-same function as SWAP

To load a second document, type ' SW ' from command mode and the current document will be switched, leaving an empty document. Load the second document in the normal way. Switching between the two documents will cause the status lines to change to suit the current document, enabling easy recognition of which document is being worked on. The status line also indicates the second document with an asterisk (*).

From edit mode, CTRL-Y performs exactly the same purpose as ' SW ' enabling very quick swapping between documents.

The CTRL-U command is extremely useful, as it enables any part of the text of dither document to be copied to the other.

To use CTRL-U, first define the block of text that is to be copied, using the block markers in the normal fashion. Type CTRL-Y to swap files, position the cursor where the block is to be copied to and type CTRL-U. If you want to remove the original text, type CTRL-Y again, fallowed by CTRL-DEL.

Another use for CTRL-U is for transferring text from one file to another - load the first file, type SWAP load the second file and use CTRL-U to copy the blocks required into the first file, before re-saving it. This is considerably quicker than using SB (save block), loading the other file and merging the saved block of text into the document.

On the CPC6128 (or 464/664 with memory expansion), each of the documents can be up to about 40k in length, and the length of one does not affect the other.

On the unexpanded CPC464 and CPC664 the combined length of both files can be up to about 40K, and they share the same memory space.

Note: the OPT command, when used, sets the options for both documents.

3. Box mode - cut and paste operations

The cut and paste facilities provided in PROTEXT only operate on continuous sections of the text. PROMERGE PLUS extends this to operate on any section of text that can be defined by drawing a rectangle. A block defined in this way is termed a 'box'. There is one new command used for box mode.

CTRL-B	:	Edit mode - Box mode toggled on or off.

To define a box, first type CTRL-B to enter box mode. The message 'BOX MODE ON ' will appear on the status line (in program mode a ' B ' will appear on the left of the status line. Move the cursor to the top left corner of the (imaginary) box and press SHIFT-COPY. Move the cursor to the bottom right of the box and press SHIFT-COPY. Markers will then appear along the left and right edges of the box. This box can now be moved, copied on deleted, using the normal block commands (CTRL-M, CTRL-COPY, CTRL-DEL). Position the cursor where the top left corner of the box is required and then use the appropriate copy/move command.

Boxes may be swapped between documents using CTRL-U. The box mode setting for the document in which the block is defined determines the method of block copying. If box mode is on, then the block will be copied as a box, otherwise it will be copied as a block of text, in normal PROTEXT fashion.

Very effective layouts can be achieved by the use of box mode. One example would be to create a page with two columns of text, along the lines of a magazine page. This is simply done by creating the document, formatting it with a width of about 35 characters and then using box mode to move the second half up to the right, positioning it alongside the first part of the text. This should be done immediately prior to printing, as any further reformatting might destroy the formatting of the two columns.

Note: When a box move is carried out and the box contains tabs, it may sometimes appear that the justification has been destroyed. The justification can be restored either by inserting an extra tab on the ruler line, or by replacing the tab with spaces.

4. New and altered printing commands

There are two new commands, in addition to those already described for PROMERGE, and a slight alteration to the PRINT and PRINTQ commands.

PRINTP	(PP)	(infile)	Print pages selectively.
PRINTPQ	(PPQ)	(infile)	Print pages selectively in NLQ mode.

PRINTP and PRINTPQ function in a similar fashion to the PRINT command described in the previous chapter for PROMERGE, with the ' Press space to print' message permitting the selective printing of pages.

PRINT, PRINTQ and their associated block commands have the ' Press space to print ' message suppressed when used with PROMERGE PLUS and the complete text, or block of text, will be sent to the printer buffer without any messages, but otherwise functions as before.

The reason for the change is that in certain circumstances the background printing facility (see below) displays a message on the screen when paper needy changing in the printer. Having the two different messages could be confusing. In the event that selective page printing is necessary, then the PRINTP and PRINTPQ commands can be used, in which case both messages may be displayed at the appropriate times, depending on the options selected.

5. Background printing

Background printing means that it is possible to continue editing another file, or even run another program, whilst text is being printed. No special commands are needed to use this feature - background printing is used automatically whenever any of the commands which direct text to the printer are used.

Background printing works in the following fashion:

When one of the print commands is used to print text, instead of the text merely being sent straight to the printer it is first put into a ' printer buffer '. This is an area of memory that is not currently being used by text or anything else.

Whenever there Is text in the buffer it will automatically be pulled out and printed, so printing starts immediately. The benefit of this feature comes from the fact that text is put into the buffer at a much faster rate than it is pulled out for printing. When all the text to be printed has been put into the buffer the command mode cursor will return and normal editing can be continued.

Printing will then continue at the same time and will not be affected by anything else (unless, of course, you do something drastic like switching the computer off !).

If a large document is being printed, it may sometimes appear that background printing is not working. This is not the case, it is simply that the amount of text to be printed is greater than the available capacity of the buffer. If the buffer is full, text is inserted into it as soon as text is removed from the other end for printing. Eventually the end of the text to be printed will be loaded into the buffer and then further editing can be carried out whilst background printing continues.

The size of the printer buffer depends on the amount of computer memory and the size of text currently in memory. On the CPC6128, part of the second 64k of memory is always used for the printer buffer, whilst an unexpanded CPC464 or CPC664 uses some of the remaining space in memory.

Up to 64k of RAM expansion may be used with the CPC464 or CPC664, providing it is compatible with the method used by the CPC6128 for switching RAM banks.

For example: With PROTEXT and PROMERGE PLUS on ROM:

On a CPC464 with a 12K file in memory the buffer is about 25K.
On a CPC464 with two 12K files in memory the buffer is about 13K.
On a CPC6128 with a 20K file in memory the buffer is about 63K.
On a CPC6128 with two 20K files in memory the buffer is about 43K.

NOTE: Whilst background printing is in operation it is inadvisable to make use of bank switching commands from other programs, as some of these do not make use of the recommended procedures, which can result in loss of data.

There is no restriction on the use of the ' SW ' or CTRL-Y and CTRL-U commands, but if the buffer is so full that there is not room for the file in the banked memory, then a message is given to this effect and the command can be re-tried when the buffer has emptied a bit.

Stopping background printing

The normal method of stopping the printer by pressing ESC will not work when background printing is in operating, since ESC is needed for other purposes, such as swapping between edit and command mode. Commands have therefore been added to allow printing to be stopped, resumed or completely abandoned. They can be used from command mode at any time during printing.

STOP	(ST)	Stop printing but leave buffer intact.
CONT	(CO)	Continue printing remaining contents of the buffer.
ABANDON	(AB)	Abandon printing completely and clear the buffer.

STOP merely pauses printing, whilst CONT causes printing to re-commence. ABANDON can be used either whilst text is being printed, in which case printing will stop and the buffer will be cleared, or after a STOP command, to clear the buffer.

If the printer is on ' STOP ' when the program attempts to send more text to the buffer, a message ' Re-start (y/n) ' will appear. ' Y ' or the RETU RN key will restart printing, whereas ' N ' will put text into the buffer, but not restart printing. Pressing ESC will cause neither to happen.

NOTE: After a STOP command, the buffer will still contain any unpainted text and ABANDON should always be used if the remainder of the text is not to be printed, as this will clear the buffer and make the space available for other purposes.

NOTE : Some printers contain printer buffers of their own, in which case printing can appear to continue after STOP or ABANDON have been used. This is because the printer's own buffer still contains text to be printed and the printer manufacturer's recommended method of clearing this buffer should then be used as well. This often consists of simply switching off the printer.

Background printing and single sheets

If printing is being done on single sheets of paper, the printer must stop after each page to allow the pager to be changed. There are two methods that can be used to do this. The first method will only work with Epson printers and some compatibles but the second method will work With all printers. The first method is to be preferred, If it works, as it permits continuous editing without interruption.

Any of the normal printing commands may be used, but if only selected pages are to be printed, the PRINTP and PRINTPQ commands should be used.

a). Method for Epson printers
(and some Epson compatibles including the Amstrad DMP 2000).

This method uses the stored command PS (paper sensing). Include the commands ' >PS ON ' and ' >CP OFF ' (single page) near the start of the text (see chapter on stored commands).

When the end of a page is reached the printer will be automatically turned off line. Simply change the paper and press the on line button on the printer and printing will continue. This operatlon does not affect any other actions the computer may be performing.

b). Method for all other printers

The '>PS ON ' stored command should **NOT** be used.

The following method should be used on all printers that are not Epson compatible. This includes some printers that are mostly Epson compatible, such as the Kaga Taxan.

When the end of a page is reached, the message:-

'Insert paper: press COPY. ESC to stop ' will appear on the screen, interrupting whatever is being done. The paper should then be changed and the COPY key pressed to restart printing, or the ESC key to stop printing.

Note: If the printer contains its own substantial printer buffer, then it may be found that the ' Insert paper ' message appears some considerable time before the end of a page is reached. It may be possible to disable this buffer and the printer manual should be consulted. It is not a fault of the program, but as a result of the printer's buffer absorbing characters to be printed, but not yet printed.
NOTE: The buffer will still contain any unpainted characters. If the remainder is not to be printed, the ABANDON command should be used to clear the buffer, for the reasons explained earlier.

6. Decimal Calculator

This is a simple calculator, called with the following command from command mode:

CALC (C)

The command always prompts for an expression. It will calculate expressions involving numbers which may have a decimal point and a fractional part, and will display the result to two decimal places. The operators available are +, -, *, and /. "Evaluation is from left to right". When the result is displayed it may be inserted into the text at the current cursor location by pressing the space bar, the program returning to edit mode. If any other key is pressed a further prompt for another expression will be issued. Press ESC to exit calculator mode and ESC again to return to edit mode.

7. File conversion

CONVERT (CV)
CONVERTB (CVB)

These commands attempt to convert an ASCII file into PROTEXT document format. CV operates on the whole file (in memory), CVB on the currently defined block.

The operations carried out are:

1. PROTEXT is switched to document mode.
2. Wherever a hard return is followed by a character that is not a space or another return, it is changed to a soft return.
3. Where a hard return is changed to a soft return as above, any spaces preceding the soft return on that line, that are followed by another space, are changed to soft spaces.
4. Characters with code number above &BF are removed.

NOTE: In most cases, where the document consists of straightforward paragraphs of text, the ' CV ' command will be entirely successful. Occasionally, where the document may contain complex or unusual layouts, such as tabulated columns of figures it may be preferable to leave these areas of text untouched, in which case ' CVB ' should be used on the remaining areas.

8. Miscellaneous commands

GOTO (G)

GOTO < line number > Command mode - the same as CTR L-G in edit mode.

Note: The line number is the actual line number in the text, not the line number displayed when in page mode (which ignores stored command lines). Using CTRL-P to toggle out of page mode will reveal the true line number.

NAME (N)

NAME < filename > Command mode - assigns a name to the current text. If the text already has a name, it will be replaced by the new name. If there is no text in memory, a single return is put into the text and the name allocated to it.

A1. SUMMARY OF COMMANDS

(a) Stored commands

The commands marked ' * ' take immediate effect, the others take effect at the next new page.

Key

a . . . an ASCII character
f . . . a filename
n . . . an integer between 0 and 255
v . . . a variable identifier
(x) . . . an optional parameter
{x}. . . a parameter that may occur 0 or more times
expr . . . a string expression
cond . . .a conditional string expression

Where more than one parameter is listed on a command line the parameters may be separated by spaces, commas, or equals signs. This is not shown in the syntax descriptions below.

*	>AV	(v) {v}	Ask for variables
	>BM	n	Bottom margin.
*	>CC	a {n} (; {n})	Define printer control code.
*	>CE	text	Centre line.
*	>CF		Close data file.
*	>CO	text	Comment line.
	>CP	ON/OFF	Continuous/single sheet printing.
*	>CS	text	Clear screen and display message.
*	>CW	n	Define microspace character width.
*	>DA		Switch printer to Danish.
*	>DF	f {f}	Define data file.
*	>DM	text	Display message.
	>EA	n	End at page.
	>EF	text	Define even footer text and turn footers on.
	>EH	text	Define even header text and turn headers on.
*	>EI		End of IF block.
*	>EL		Else - print block if previous IF condition false.
*	>EM	n	Even side margin.
*	>EN		Switch printer to English.
*	>EP	(n)	Even page throw (can be conditional).
*	>EX	text	External command.
	>FF	ON/OFF	Form feeds enabled/disabled.
	>FM	n	Footer margin.
	>FO	text	Define footer text and turn footers on.
	>FO	ON/OFF	Turn footers on/off .
*	>FP	ON/OFF	Formatting whilst printing on/off.
*	>FR		Switch printer to French.
*	>GE		Switch printer to German.
	>HE	text	Define header text and turn headers on.
	>HE	ON/OFF	Turn headers on/off.
	>HM	n	Header margin.
*	>ID	v	Print block if variable defined.
*	>IF	cond	Print block if condition true.
*	>IN	file	Insert file

*	>IT		Switch printer to Italian.
*	>IU	v	Print block if variable undefined.
*	>LS	n	Line spacing.
*	>MC	n {n}	Define microspace code sequence.
*	>MS	ON/OFF	Microspacing on/off.
	>NC	n	Number of copies.
	>NP	ON/OFF	Enable/disable new page at end of printing.
*	>OC	n {n}	Output codes to printer.
	>OF	text	Define odd footer text and turn footers on.
	>OH	text	Define odd header text and turn headers on.
*	>OM	n	Odd side margin.
*	>OP	(n)	Odd page throw (can be conditional) .
*	>PA	(n)	Page throw (can be conditional).
	>PL	n	Page length.
	>PN	nn	Page number.
*	>PR	filename	Load printer driver.
*	>PS	ON/OFF	Paper sensing on/off .
*	>RC	a {n}	Redefine character.
*	>RJ	ON/OFF	Right justifying on/off .
*	>RP		Repeat.
*	>RU	v {v}	Read variables unconditionally.
*	>RV	v {v}	Read variables and pad with nulls.
	>SA	n	Start at page.
*	>SK	cond	Skip printing if condition true.
*	>SM	n	Side margin.
*	>SP		Switch printer to Spanish.
*	>ST	text	Stop printing and display text.
*	>SV	v=expr	Set variables.
*	>SW		Switch printer to Swedish.
	>TM	n	Top margin.
*	>UN	cond	Until condition is true.
*	>WT	text	Wait and display text.
	>ZM		Zero margins.
*	>>>	text	Comment line.

If column one contains '>', and columns two and three each contain one of: space, 'L', '-', or '!' then the line is a ruler line.

(b) New commands and extensions to existing PROTEXT commands

Extended PROTEXT commands

HELP	(H)		- Extended Help menu.
PRINT	(P)	P	(input filename)
PRINTF	(PF)	PF	(output file) (input filename)
PRINTS	(PS)	PS	(input filename)
VERSION	(VER)	VER	- Lists version numbers of PROTEXT and PROMERGE

PRINT, PRINTS and PRINTF work in the normal PROTEXT fashion if used without the optional filename parameter and the text in memory will be printed. If an input filename is specified, the file will be printed from disc. It is not possible to use the Block options when printing from a file.

Note: PRINT has a slightly different action in PROMERGE PLUS.
See notes in PROMERGE PLUS chapter on background printing.

New commands

OPTION	(OPT)	OPT 0	Reset to defaults	(no bits set)
		OPT 1	Box mode on	(set bit 0)
		OPT 4	Help on	(set bit 1)
		OPT 8	Page mode off	(set bit 2)
		OPT 16	Right-justify off	(set bit 4)
		OPT 32	Word-wrap off	(set bit 5)
		OPT 64	Overwrite mode on	(set bit 6)
		OPT 128	Program mode	(set bit 7)
		OPT 256	Tabs and CRs on	(set bit 8)
		OPT 512	CTRL-V on	(set bit 9)
		OPT 1024	Spaces on	(set bit 10)

PRINTQ	(PQ)	PQ (input filename)	- Print, NLQ mode.
PRINTQB	(PQB)	PQB	- Print block, NLQ.
SPLIT		SPLIT <input filename> (max size bytes)	- Split file.
TYPEWRITER	(TW)	TW (integer)	- Typewriter mode.

(c) External commands

External commands are commands that can be called from another program, usually Basic and are prefixed with a bar symbol (|).

PROTEXT	(P)	Enter PROTEXT.
PRINTER		Loads a printer driver.
ROMON7		Turns off all ROMs except the AMSDOS and Basic ROMS.

NOTE: PROTEXT (P) commands have been extended to permit the use of any PROTEXT command from Basic. e.g. |P, "L filename ".

(d) Additional commands PROMERGE PLUS only

Command mode

ABANDON	(AB)		Abandon printing completely and clear buffer.
CALC	(C)		Calculator.
CONT	(CO)		Continue printing remaining contents of buffer.
CONVERTB	(CVB)		Convert block to PROTEXT format.
GOTO	(G)	< nn >	Goto line number.
NAME	(N)	< name >	Assign name to current file.
PRINTP	(PP)	(infile)	Print pages selectively.
PRINTPQ	(PPQ)	(infile)	Print pages selectively in NLQ mode.
SPLIT		< file > (nn)	Split large file into smaller files.
STOP	(ST)		Stop printing but leave buffer intact.
SWAP	(SW)		Swap between two documents in memory.

Edit mode

CTRL-B	Box mode toggled on or off.
CTRL-U	Copy block from the other document.
CTRL-Y	Same function as SWAP.

A2. TECHNICAL SUMMARY OF PROMERGE STORED COMMANDS

This chapter contains full syntax descriptions of all the new or extended stored commands. The PROTEXT manual should be consulted for details of any commands not covered in this section.

STORED COMMANDS

AV	Ask for variable from keyboard.
Syntax:	>AV (< string >) < identifier > (< small integer >) { (< string >) < identifier > (< small integer >) }
Examples:	>AV " Name " name, 20, '"Address line 1" addr1 >AV name, addr1 , addr2, addr3, addr4
CC	Redefine control code.
Syntax:	>CC < letter > (< code >) (;{ < code > }) >CC @
CE	Centre line.
Syntax:	>CE < text >
CF	Close file.
Syntax:	>CF
CP	Select continuous printing or single sheets.
Syntax:	>CP ON >CP OFF
CS	Clear screen.
Syntax:	>CS < message >
CW	Set character width for microspacing.
Syntax :	>CW < small integer >
Examples:	>CW 10 (elite characters) >CW 7 (condensed) >CW 14 (condensed enlarged)
DA	Danish.
Syntax:	>DA
Note:	The sequence ESC "R" 4 is printed. The control code f is defined as 27, "R", 3; 27, "R", 4 The control code g is defined as 27 "R", 2; 27, "R", 4 £ is defined as 27, "R", 3, "#", 27, "R", 4 # is defined as "#"
DF	Define data file or files.
Syntax:	>DF < filename > (< filename >)
Example:	>DF "data" >DF names1, names2, names3
Notes:	It is not possible to use DF and IN at the same time. If more than one DF command is required, a CF command must appear in between.
DM	Display message.
Syntax:	>DM < message >

EA	End at page number.
Syntax:	>EA <integer>
EI	Endif. Matches with ID, IF or IU.
Syntax:	>EI
EL	Else. Matches with ID, IF, or IU.
Syntax:	>EL
Note:	IF blocks may be nested within EL blocks.
EN	English.
Syntax:	>EN
Note:	Switch printer to English. Epson-compatible printers only. The sequence ESC "R" 3 is printed. The control code f is defined as 27, "R", 1; 27, "R", 3 The control code g is defined as 27 "R", 2; 27, "R", 3 £ is defined as "#" # is defined as 27, "R", 0, "#", 27, "R", 3
FF	Enable or disable the use of form feed codes.
Syntax:	>FF ON >FF OFF
FP	Enable or disable formatting whilst printing.
Syntax:	>FP ON >FP OFF
FR	French.
Syntax	>FR
Note:	The sequence ESC "R" 1 is printed. The control code f is defined as 27, "R", 3 : 27, "R", 1 The control code g is defined as 27, "R", 2: 27, "R", 1 £ is defined as 27, "R", 3, "#", 27, "R", 1 # is defined as "#"
GE	German.
Syntax	>GE
Note:	The sequence ESC "R" 2 is printed. The control code f is defined as 27, "R", 3 : 27, "R", 2 The control code g is defined as 27, "R", 1: 27, "R", 2 £ is defined as 27, "R", 3, "#", 27, "R", 2 # is defined as "#"
ID	If defined. Matches with EI.
Syntax:	>ID <identifier>
IF	Print if condition true. Matches with EI.
Syntax:	>IF <condition>
Note:	IF blocks may be nested to a depth of 7.
IN	Insert file.
Syntax:	>IN <filename>
Note:	It is not possible to use IN and DF at the same time.

IT	Italian.
Syntax	>IT
Note:	The sequence ESC "R" 6 is printed. The control code f is defined as 27, "R", 3 : 27, "R", 6 The control code g is defined as 27, "R", 2: 27, "R", 6 £ is defined as 27, "R", 3, "#", 27, "R", 6 # is defined as "#"
IU	If undefined. Matches with EI.
Syntax:	>IU <identifier>
MC	Set microspace character code sequence.
Syntax:	>MC <code> {<code>}
Example:	>MC 27, "L", 1, 0, 0
MS	Enable or disable microspacing.
Syntax:	>MS ON >MS OFF
NC	Number of copies.
Syntax:	>NC <small integer>
Note:	The number of copies is ignored if a data file is defined.
NP	Enable or disable new page after printing.
Syntax:	>NP ON >NP OFF
OC	Output codes to printer.
Syntax:	>OC < code > (< code >)
Examples:	>OC 27, "3", 12 >OC 27, J, 20
PR	Load printer driver.
Syntax:	>PR < filename >
PS	Enable or disable paper sensing.
Syntax:	>PS ON >PS OFF
Note:	On Promerge Plus, this suppresses the 'Insert paper ; press COPY ' message .
RC	Redefine character.
Syntax:	>RC < character > = < code > { < code > }.
Examples:	RC "!" = "!" #08 "." RC £ 27 ">" 1 27 "="
RJ	Enable or disable right justification.
Syntax:	>RJ ON >RJ OFF
RP	Repeat. Matches with UN.
Syntax:	>RP
Note:	Repeat loops may not be nested.

RU — Read variable unconditionally.
Syntax: Are < identifier > { < identifier > }

RV — Read variable from data file.
Syntax: >RV < identifier > { < identifier > }
Example: RV name, add1l, addr2, addr3, addr4, addr5, dummy

SA — Start at gage number.
Syntax: >SA < Integer >

SK — Skip copy on condition true.
Syntax: >SK < condition >

SP — Spanish.
syntax: >SP
Note: The sequence ESC "R" 7 is printed.
The control code f is defined as 27, "R", 3 : 27, "R", 7
The control code g is defined as 27, "R", 2: 27, "R", 7
£ is defined as 27, "R", 3, "#", 27, "R", 7
is defined as "#"

ST — Stop printing.
Syntax: >ST < message >

SV — Set variable.
Syntax: >SV < identifier > = < expression > (< identifier > = < expr >).
Example: >SV name "Peter Hill" sex= "M"

SW — Swedish.
Syntax: >SW
Note: The sequence ESC "R" 5 is printed.
The control code f is defined as 27, "R", 3 : 27, "R", 5
The control code g is defined as 27, "R", 2: 27, "R", 5
£ is defined as 27, "R", 3, "#", 27, "R", 5
is defined as "#"

UN — Until . Matches with RP.
Syntax: >UN < condition >

ZM — Zero margins.
Syntax: >ZM.

KEY TO SYNTAX

(. . .)	an item in parentheses is optional.
{ . . . }	an item in curly brackets may occur any number of times (possibly zero)
<character>	an ASCII character, optionally in quotes.
<string>	a sequence of ASCII characters, enclosed in quoted. It may include variable references.
<message>	a string, except that the quotes may be omitted.
<filename>	a legal file identifier under the active filing system.
<small integer>	an integer between 0 and 255.
<identifier>	a variable identifier. Must start with a letter and may contain letters, digits, ".", "-", and "?".
<code>	An ASCII code. The codes may be specified as decimal numbers, ex numbers (prefix with & or #) or ASCII characters. If an item is not a number it is sent as an ASCII character. To send the ASCII code of a decimal digit enclose the digit in quotes. Codes may be separated by commas or spaces.

< expression >	is < num expr > or < str expr >
< num expr >	is < num item > (< num op > < num expr >)
< num op >	is +, -, * or /
< num item >	is < fixed pt no. > or < identifier > with value a fixed pt no.
< fixed pt no. >	is (< sign >) < uint > (. (< uint >)) or . < uint >
< uint >	is unsigned integer
< str expr >	is < str item > (+ < str expr >)
< str item >	is < num item > or < id > [(W) < int >] or < id > [(W) < int > :] or < id > [(W) < int > : < int >] or < id > [W-1]
< int >	is integer less than 256
< condition >	is < str cond > or < num cond >
< str cond >	is < str item > < str comp > < str item >
< str comp >	is one of : =, <, <=, >, >=, < >, IN, NOTIN
< num cond >	is < num exp: > < num comp > < num item >
< num comp >	is one of : =, <, <=, >, >=, < >

separators — Where more than one parameter is listed on a command line the parameters may be separated by spaces, commas, or equals signs. This is not shown in the syntax descriptions.

Variable references :

To print the value of a variable at any point in the text, type the variable name with an "&" immediately before and after it, e.g. &name&. If the value of the variable is a null string, and there is nothing else on the line, then the line is omitted altogether. If the value of the variable is the null string and there is a space after the variable reference then the space is not printed.
If the blank line or space Is required then type " ! " before and after the variable name, e.g. !name! .

A3. MAIL MERGE - MASTERFILE 464 + EXTENSIONS & MASTERFILE 128

PROMERGE can readily make use of data files created either by Masterfile Extension Programs with the 464 versions or the Export facility of Masterfile 128 or Masterfile 3.
Masterfile is copyright of Amsoft and Campbell Systems.

There are three likely situations that may be encountered, which will be discussed separately:

1. Creating a Masterfile database for use with PROMERGE.
2. Utilising an existing Masterfile database for use with PROMERGE.
3. Utilising existing data f iles created by Masterfile Export facilities.

1. Creating a new Masterfile database.

If a database is being specially created, with the intention of using it with PROMERGE, then there are a number of things which can be done to make creation of suitable data files as easy as possible

.

a) . When creating fields for addresses, use a separate field for each line of the address, rather than make use of the ' Line-break ' control code which Masterfile permits.

b). Avoid the use of punctuation marks in the data fields if possible.

c) . Avoid using quotation marks in the data fields if possible.

If a database is created following the above suggestions, then creation of suitable data files is entirely straightforward and irrespective of whether the ' Export ' option of Masterfile 128 is used, or the 'MASTEXPT ' program from Masterfile Extension Programs, the following procedure can be followed:-

1. Select the records required f or transfer.
2. Use the Export program (Option in 1 28 version) and answer the prompts as follows :-

Data identifiers	Y/N	N	
LF as well as CR	Y/N	Y	
Edit line breaks	Y/N	Y	(There should be none to edit)
Soft EOF	Y/N	Y	

In addition 128 users should specify the following :-

Comma delimited	Y/N	N

3. Field references should then be entered on request. The order is not important. Unless you wish to enter some text as a default text, the dollar sign ' $ ' should be specified as the default for all fields. It is undesirable to leave defaults as empty strings.

The data file created by this method will be suitable for use with PROMERGE, using either ' RU ' or ' RV '

2. Utilising an existing Masterfile database.

There are three possible situations which can cause problems when utilising data from an existing database:-

a). Fields containing the Masterfile ' line break ' feature have been used, which may result in a varying number of data fields in the data merge file.
b) . Fields contain commas and/or quotation marks.
c). Both of the above situations are present in the file.

If none of the above apply, then the data file can be created in exactly the way described above. If the situation in a) . or c) . above is true. then the problem can be minimised by making sure that the Masterfield field which contains the ' line break ' feature is the last one specified when creating the ' export file '.

The file should be created in exactly the same way as already described, but if the fields contain commas or quotation murks it will be necessary to do some further processing before the file is suitable for use with PROMERGE.

This is easily accomplished by loading the newly created file into PROTEXT and making use of the REPLACE option in PROTEXT to ' wrap ' each field in quotes, before saving the modified file. The procedure is straightforward and only takes a short time to carry out.

a) . It is necessary, to know that any quotation marks in the fields are all of one type, so the following should be typed in from command mode:- R ' " ' " ' " GA followed by pressing the RETURN key. This will have the effect of automatically replacing all double quotes throughout the file with single quotes.
b) . AII fields can be wrapped In quotes by the following replace sequence: R ! . ' " ! . " ' GA followed by pressing the RETURN key. This will automatically find every carriage return (CR) and insert a set of quotes before and after each one.
c). Finally it is necessary to remove the surplus quotes at the end of the file and using CTRL- [, go to the start of the file and insert quotes at the very beginning.
d) . Re-save the data file.

The above sequence will create a data file which is entirely suitable for use with PROMERGE. If the records created could be of variable length, due to the use of the Masterfile ' line break ', when creating the merging document the 'RV' command should be used. If all records are the same length, then either ' RU ' or ' RV ' can be used to read the data.

NOTE: Users of Masterfile 128 may find the following sequence for wrapping fields preferable.

1 . When using the Export option, specifiy Yes in answer to the comma delimiter question, which will automatically wrap the fields and correct any problems with the types of quotes.
2. Load the data file into Pretext and us) the following from command mode:R ! . ! . ! . GA which will have the effect of inserting the required blank line between each record.
3. Resave the file.

3. Utilising existing data files.

If the data files are of the sort created by the above methods, then they can be used in exactly the same way, but Masterfile also permits the creation of 'export' files which make the use of identifiers.

If any fields created by the ' line break ' are the last fields in the record, then it is sufficient to use PROTEXT'S R EPLACE option to search for '&? ' and replace it with a null by using R &? " " GA which will simply remove all occurrences of ' & ' and the following character. ' RV ' should then be used to read the data.

If a data file contains the identifiers and ' line break ' created fields, but these are not at the end of a record, there is still a way to create a suitable file. In this case PROMERGE will be used to create the new file. Create a text file along the lines of the following, modifying it to suit the format of the data file to be converted and then use the PRINTF option, which will save the converted file to disc as an Ascii file suitable for use by PROMERGE.

```
>ZM                 ; zero all margins
>PL 7               ; page length equal to number of fields inc. dummy
>DF datamerg        ; name of file to convert
>RV name add1 add2 add3 add4 tel dummy          ; variables to read
>IF add4 [1:2] = "&T"                  ; if add4 is tel field
>SV tel=add4 [3:] add4 "$"             ; then tel = add4. (stripped of identifier)
>EI                                    ; and add4 = $
>SV name=name [ 3: ] add1=add1 [ 3: ]      ; strip name and add1
>IF tel [ 1 : 2 ] = "T"                    ; and if tel has identifier
>SV tel=tel  [ 3: ]                       ; then strip it as well
>EI
! name !
! add1 !
! add2 !
! add3 I
! add4 !
! tel !
! dummy !
```

The above example is based on the assumption that the data file to be converted contains fields marked with identifiers and in the order: - name, three or four address lines and finally a field for telephone. The conversion should be carried out using PRINTF and will be completely automatic. The result will be a file containing a constant number of fields in each record, any absent fiends being marked with the ' $ ' marker. ' RV ' or ' RU ' may be used to read the data.

With a bit of thought, it should be possible to convert virtually any sort of Masterfile ' export ' file into a suitable format, as well as utilise the Export facilities of Masterfile to create suitable files from either new or existing database records.

```
>CO     Example routine to select one of four data files.
>CO     no data file required
>AV     bad name
>SV     suff=" junk s - z l - r e - k a - d "
>RP
>AV     suff=suff [ w2: ]
>UN     name [ 1 ] > suff [ 1 ]
>SV     df = " address " +suff [ w1 ]
>DF     &df&
```

This file makes use of a RP UN loop to repeatedly shorten a variable until the first letter of ' name ' is greater than or equal to the first letter of the variable. ' junk ' is simply there to be discarded on the first loop. Finally, joining of variables and text is used, followed by making use of the ability to use a variable in a command, to define the data file. This would be a suitable routine to select a data file to be read, according to the surname, enabling data to be kept in a number of files.

```
>CO     using stored commands to set other stored commands.
>CO     no data file
>AV     " Side margin " sm
>AV     " Page length " pl
>AV     " Single sheet Y/N " cp
>AV     " No of copies " nc
>IF     " Y " IN cp
>SV     cp= "OFF"
>EL
>SV     cp= "ON"
>EI
>SM     &sm&
>PLI    &pl&
>CP     &cp&
>NC     &nc&
```

This one simply asks for a number of details about the options required and then uses the variables in the appropriate stored commands.

```
>CO      Invoice printer and calculator
>CO      No data file required
>AV      vatrate=0.15
Description         Retail    Quant    Disc      Net       VAT       TOTAL
>RP
>AV " Description of goods " descrip 12
>AV " Retail price " ret  "Quantity" qty"' "Discount" disc
>IF disc=" "
>SV disc=0
>EI
>SV      totret=ret*qty off=totret/100*disc net=totret-off vat=net*vatrate
>SV      totinc=net+vat gnet=gnet+net gvat=gvat+vat gtotinc=gtotinc+totinc
>-----------------------------.--------------.----.--------------.-----.--------------------.-------.-----R
&descrip&              &ret&      &qty& &disc&   &net&    &vat&              &totinc&
> AV "Another entry? Y/N " yeno
>UN yeno [ 1 ] < > " Y "

                                                    ---------------------------------------------
                                          TOTALS   &gnet&  &gvat&           &totinc&
                                                    ---------------------------------------------
```

The above template file creates invoices. Based on information requested, it will calculate the various totals, net values and VAT. After each item the option of entering another item is offered. Any answer but ' Y ' will stop further entry and the grand totals will be printed. Use the PRINT command.

```
>>>      This example is part of an invoice generating template
>>>      no data f le is required
>RP       ; repeat enter order until correct
>AV "Pretext order (Eprom, disc, cassette) : " orders
>SV orderp=orderp + " 0 0 0 "                    ; default quantises of zero
>IF orders[I 1 ] = " "                           ; if no figures were entered
>SV orderp=orderp [ 2: ]                         ; strip leading space
>EI
>SV ep=orderp [ w1 ] dp=orders [ w2 ] cp=orderp [ w3 ]
>AV "Re-enter order (y/n?) ? ", again ?
>UN again ? [ 1]) < > " Y "
```

This is an extract from another type of invoice generator. in the full template, the prices would be calculated from the quantities and the procedure repeated for other products. The prices would be stored in the template file as variables.

The quantities of each of the 3 formats of Protext are entered as a single string, e.g. ' 10 20 10 ', and this string is split into the 3 individual numbers. Each quantity defaults to zero if omitted. The whole ls enclosed in a repeat loop to allow the order to be re-entered if a mistake was made.

```
>CO Data file creator.
>CO No data file required
ZM                                      ; set margins to zero
>CS Type END to stop
>AV "Enter name " name                  ; ask for name
>IF name= "end"                         ; see if it equals end
>ST                                     ; and if so stop
>EI
>AV " Telephone number " tel "Address 1 " addr1 " Address 2 " addr2
>AV " Address 3 " addr3 " Address 4 " addr4
>SV dummy=" "
>IF " ' " IN addr1                      ; check for double quotes in first address
>SV addr1=" ' " + addr1+ " ' "          ; if so wrap address in single quotes.
>EL                                     ; or
>SV addr1 =' " ' +addr1+ ' " '          ; wrap address in double quotes
>EI
" ! name ! "
" ! tel ! "
! addr1 !
" ! addr2 ! "
" ! addr3 ! "
" ! addr4 ! "
! dummy !
>IN datamake                            ; call file again .
```

The above file will create a data file of names, telephone numbers and addresses. The file must be saved on disc with the same name as used in the ' >IN ' command at the end. Typing ' end ' in answer to ' Enter name ' will close the file and stop further processing. The ' IF ' command checks for house names with double quotes and if found, wraps the data in single quotes. The remainder of the variables ate wrapped in quotes at the printing stage. The last line calls the file from disc again. The file is created by usingPRINTF with a filename.

```
>CO routine to separate off initials
>CO no data file required
>AV name                                ; Request name to separate initials from.
>SV n1=name [w1] m2=name [ w2] n3=name [w3] n4=name [w4]
>IF n4=name [w-1]                       ; check for same as last word in ' name '
>SV inits=n1+n2+n3                      ; if so, 'inits' = n1 , n2 and n3
>EI
>IF n3=name [w-1]                       ; repeat for third word
>SV inits=n1+n2                         ; if so, 'inits' = n1 and n2
>EI
>IF n2=name [w-1]                       ; repeat for  second word
>SV inits=n1                            ; if so, 'inits' = n1
>EI
&inits&   &name&
```

This file is an example of manipulating variables, in this case to extract the initials in a person's name. It takes account of likely variations in the number of initials.

A4. SAMPLE PROMERGE DOCUMENTS

This chapter consists of a number of sample PROMERGE documents intended to give some idea of the variety of possible uses for PROMERGE. Some of them are complete documents in their own right, but many are working extracts from larger documents, intended to demonstrate solutions to a variety of problems.

The first comment line in each program describes the use of the document and the next line describes the type of records required in the data file. There is a brief description of how the program works and what it does, after each document.

The last example is worthy of study, not so much for its usefulness, but as an example of methods of programming and the versatility of PROMERGE.

The first example is only for reference and is 'EXAMPLE6' from the earlier chapters, listed in full.

```
>CO      EXAMPLE6
>CO data file requires name, tel, 4 address lines
>PL 24              ; for screen display
>CP OFF             ; purposes only
>DF datfile3        ; open data file
>RU name tel addr1 addr2 addr3 addr4 dummy        ; read data fields
>SK addr4 > " "             ; check for last address line empty. If so, skip
>AV "Enter Date " date      ; request date
>SV year = "1986"           ; set the year

&name&
&addr1&
&addr2&
&addr3&
&addr4&

Dear &name&
>CE      Renewal date - &date& &year&
                Thank you for your letter about the insurance for &addr1&
                &addr2& &addr3& &addr4&. We think that you will find the rated
                quoted . . .

>IF tel > " "                       ; if telephone - print
                If you would be so kind enough to telephone us . . .
>EL                                 ; otherwise - print
                If you would be kind enough to write us . . .
>EI                                 ; end of IF EL block
```

```
>CO                                                  Prime number generator
>DF primes ; never accessed so any filename on disc can be used
>ZM
>IU i
>SV i=3.00
>EL
>SV i=i+2
>EI
>SV j=3.00
>>>
>RP
>SV t=i/j
>SK t[ w - 1 ] = 0              ; if remainder 0
>IF j*j > i
>SV iint=i[ w1 ]
&iint&
>SK 1=1
>EL
>SV j=j+2
>EI
>UN 0=1
```

This file calculates prime numbers and should be started with PS.
The only point to note is the ' DF ' command, which must have the name of a file on disc even though it is not actually accessed. It is necessary, as it permits ' SK ' to work.

This manual was brought to you by **ROBCFG**

Original scans by **CPCManiaco**

Typesetting by Whitgift Business Services Ltd. The Studio, Ledbury Place. Croydon. 01-680-7997