PANDAAL
software

PANDAAL MARKETING, 22 Linford Forum, Rockingham Drive
Linford Wood, Milton Keynes, MK14 6LY

MANUAL

KEMPSTON

Mouse Software

FOR ZX SPECTRUM

SYSTEM REQUIREMENTS:

Spectrum 48K, Spectrum +, Spectrum 128K, Spectrum +2, Spectrum +3

KEMPSTON DATA LTD 22 Linford Forum, Rockingham Drive
Linford Wood, Milton Keynes MK14 6LY, England

Tel: 0908 677886 Telex: 82304 FORUM G Fax: 0908 676066

CONTENTS

1.0 INSTALLING YOUR KEMPSTON MOUSE

The **Kempston Mouse** is designed to function with the whole range of Spectrum computers: Spectrum 48K, Spectrum +, Spectrum 128K, Spectrum +2 and Spectrum +3.

1.1 CONNECTING THE INTERFACE

The interface should be plugged into the expansion port at the rear of the Spectrum BEFORE applying the power. Under no circumstances should you attempt to insert or remove the interface whilst the power is on, otherwise permanent damage may occur to both the interface and the computer. The interface may come with an additional cable, to allow connection to the Spectrum +3. In either case the interface or cable should be pushed firmly into place. If there is any undue resitance, check that the edge connector and Spectrum edge card are aligned correctly. The mouse plugs directly into the 9-way D connector on the interface. The power can now be turned on.

1.2 HOW TO USE THE MOUSE

The mouse is primarily a 'pointing' device. It can be used in the same way as the keyboard cursor keys or a joystick to move a pointer about the screen. To use the mouse it should be placed on a clean, flat surface. The mouse should have its 'tail' running away from your hand, with the buttons at your fingertips, and the main body of the mouse under the palm of your hand. It is recommended that the mouse is not used on a surface that may result in debris being picked up and rolled into the mechanism, as this may seriously affect its performance. Mouse pads provide an effective anti-static surface but are not essential for satisfactory operation.

The mouse can be moved around effecting movements of a cursor on the screen with the appropriate software. The mouse will generally work more satisfactorily at slower speeds, but by experimenting you should find it easy to control a pointer on the screen.

The two buttons are independent, their function being determined by the software used with the mouse.

1.3 TESTING THE MOUSE

The mouse has been designed and manufactured to be trouble free. We have included a simple mouse test routine on the cassette, to enable you to verify that the mouse is working correctly. The test program can be loaded from either 48K or 128K BASIC. The test routine should be the first program on the cassette.

To load the test program enter:

LOAD **"Tester"** and play the tape

Once loaded the tape should be stopped.

The mouse test routine has been written in BASIC, using the Kempston Mouse Toolkit which accompanies the mouse. At this stage some of the commands in the program may seem unfamiliar to you. Full details of the command structure, and how to use the toolkit are included in Chapter 2 and the Appendices of the manual.

The program should autorun, and the screen will clear with a message:

KEMPSTON MOUSE TESTER V X.X

and place an arrow in the centre of the screen. You should be able to move the arrow around the screen by moving the mouse. Clicking the left and right buttons should put an 'L' and 'R' on the screen.

If the mouse does not appear to function then check that you have installed the interface correctly. If you can find no faults then consult your supplier for advice.

To return to BASIC press [BREAK] and a mouse button.

1.4 CARE OF YOUR MOUSE

The mouse itself has a silicone rubber coated ball which may need occasional cleaning. The ball can be removed by turning the ball cover clockwise and turning the ball out. Once cleaned the ball should be returned and the cover locked. We recommend that you do NOT take the main housing apart for any reason, as there are no user-serviceable parts inside the mouse.

2.0 KEMPSTON MOUSE TOOLKIT

The **Kempston Mouse Toolkit** provides a simple set of instructions to control and monitor the Kempston mouse, accessible from BASIC. The software is based around the revolutionary **WIMP** (Windows, Icons, Mouse, Pointers) program control method. The toolkit enables the user to patch routines into his/her program and produce highly professional results. The Icon/Pointer designer supplied with the toolkit provides an example of the type of result that can be achieved with the simple to use commands.

2.1 WHAT IS WIMP?

The **WIMP** (Windows, Icons, Mouse, Pointers) program control method was designed to provide a user-friendly interface for computers. The system is based on an icon (symbol) based environment, with the mouse used as a means of selection. The mouse was developed as an alternative to the computer keyboard. 'Clicking' a button on the mouse when pointing to an icon provides the method of selection. With the appropriate software the keyboard becomes almost redundant. If you are not familar with this programming philosophy then the Icon/Pointer Editor and Demonstration program supplied on your cassette provides suitable examples.

Windows are user definable areas of the screen which can be opened, and printed to, without affecting other parts of the screen. They provide a means of over-printing on the screen, the data below the window being recoverable when the window is closed down. In addition to simple text or graphic windows, windows can be constructed in such a way as to provide a 'menu' of selectable options. These can then be allowed to 'pop-up' or be 'pulled-down' anywhere on the screen.
Icons are graphic symbols that can be placed at any location on the screen, and are often used as a means of representing options more effectively than simple text would.
The Mouse and Pointer provide the method of input. The on screen pointer can be controlled by the mouse. 'Clicking' one of two(in the case of the Kempston mouse) or in some cases three buttons, provides a means of performing some pre-determined function.
When combined the user is then only confronted by a series of icons, 'pop-up' or 'pull-down' menus, and a pointer. Simplicity!

The Kempston Mouse Toolkit provides all the routines needed to add a 'WIMP' environment to your own programs.

2.2 LOADING THE CASSETTE

The cassette supplied should contain four programs: a mouse test program, a toolkit demonstration written in BASIC, an icon/pointer editor and the toolkit code.
To load each program insert the Toolkit cassette and type LOAD " Prog_name", press ENTER and start the cassette player running.
The programs you should find on the cassette are **Tester**, **Demo**, **IconEdit** and **Toolkit**. Each program should load with a copyright screen.
We suggest that before you try programming with the toolkit you look at the demonstration program, **Demo**, a full listing of which can be found in Appendix E.

2.3 TOOLKIT

The following sections of the manual are concerned with the ideas behind the toolkit, and the method of implementing these ideas in your own BASIC programs.

2.3.1 SETTING UP A WINDOW

A **window** is simply an area of screen display in the shape of a rectangle. However it has several important properties; the most important being that each one has a number associated with it, so that it can be easily recognised when it is clicked. '**Clicking**' is the process of using the mouse to move the on screen pointer over the window and pressing one of the buttons on the mouse to select that window. When a window is set up, it usually first clears the area it occupies allowing for the printing of text or graphics in the window. However, the data that was cleared is not lost. On removing the window the screen display is restored. So, how is a window set up?

The size and shape of each window is stored in memory in the form of a table. Unfortunately, the link between machine code and BASIC on the Spectrum is a weak one. So, to use the Toolkit from BASIC it is necessary to use the untidy method of **POKE**ing values into memory. If you are not familiar with this, the instruction has the form:

POKE M,V

where M is an address in memory,in this case somewhere in the table, and V is the value to be put into the table. All of the addresses used are given in Appendix A. To prevent confusion all locations in the toolkit address table have labels associated with them, and are so referenced for the remainder of this manual.

How then is a window defined? Consider placing a light blue coloured window, with a dark blue foreground in the centre of the display. First, it is necessary to define the co-ordinate system for the screen. The screen is divided into a grid with 32 squares across (the X position), and 24 squares down (the Y position). All points are referenced relative to the square in the top left hand corner. So the position 4,7 means four squares to the right of the left hand edge of the screen, and seven squares from the top.

Back to defining a window. If the size of the window is 4 by 6 (4 squares across and six down), then to position the window in the centre of the display:

XPOS = (32 - **XSIZE**)/2 - 1 = 13 **XSIZE** = 4
YPOS = (24 - **YSIZE**)/2 - 1 = 8 **YSIZE** = 6

To insert these values into the toolkit address table simply:

POKE **XPOS**, 4 : POKE **XSIZE**, 8
POKE **YPOS**, 6 : POKE **YSIZE**, 8

The labels **XPOS**, **YPOS** etc are defined in Appendix A, and refer to entries in the toolkit address table. So the statement POKE XPOS,4 actually means:

POKE 61681, 4

Having defined the position and the size of the window, how is the colour specified? The window is to have a light blue coloured background with a dark blue foreground. The Spectrum colours are numbered 0 (Black) to 7 (White). If you are not familiar with using colours on the Spectrum then we suggest you refer to the appropriate section of the Spectrum User Guide.
Light blue has a value 5 and dark blue 1. To calculate the toolkit colour parameter, it is necessary to take the foreground colour and add 8 x the background colour. In this case:

COL = 1 + 8 * 5 = 41

The window can be given extra bright or flash attributes by adding a further 64 or 128 respectively to **COL**. To store this value:

POKE **COL**, 41

How then does the toolkit use these parameters to produce the required window? The toolkit has a BASIC command to invoke the function. Appendix B gives a list of all the memory addresses of the toolkit commands, and the labels associated with them.

The Toolkit command structure is:

LET L = USR *memory address*

In this way, when the instruction is executed the BASIC variable L will contain an error. These are listed in Appendix C, and will be explained further on in this manual.

The command used to set up a window is:

LET L = USR **SETUP**

However, prior to invoking this command it is necessary to add:

POKE **TYPE, 1**

This is explained in section 2.3.2. If the sequence of commands above have been entered correctly, then a window should appear in the centre of the display. The Toolkit can support up to 16 windows.

The Toolkit has also done some work behind the scenes, assigning a number to the window so that it can be recognised. The manner in which this is achieved is quite simple. The new window is one more than the number of currently defined windows. The command used to find the number of defined windows is:

LET **L** = PEEK **NOW**

PEEK is the opposite of the POKE command. The PEEK **NOW** command is used to note how many windows are currently defined. This also happens to be the number of the last window defined. The numbering of windows is a useful part of the Toolkits' housekeeping function, as will be explained later.

2.3.2 DIFFERENT TYPES OF WINDOW

Thus far, the only type of window that has been introduced is one with a plain area. The Toolkit, however, supports four different types of window. The only difference between them, is the way they appear on the screen.

The first type (**type 0**) is probably the most useful, but also the most complicated. The area is cleared and filled with a background colour, and a border is drawn around it. Text is then placed in this window. The procedure and format of the text is explained later.

The second type (**type 1**) is that already described in section 2.3.1. The area is cleared and filled with a specified background colour.

The third type(**type 2**) does neither of these and draws nothing on the screen, it is invisible. The purpose of this type of window is to allow the user to draw graphics in the window instead of text, whilst not wanting to remove what is already on the screen within the window area.

The fourth, and final type, (**type 3**) goes more than half way to what type 2 is trying to achieve. It can only be used with windows of size 2 by 2, it is in fact an **Icon.** This is in fact a graphic symbol which can often represent an option more effectively than simple text. The Toolkit can handle eight different icons simultaneously. Definition of icons is achieved using the Icon/Pointer Editor **IconEdit** (see section 2.3.6). To select an icon the command:

POKE **ICON**, *Icon Number*

is invoked. In all other respects the different types of window behave in exactly the same way.

The type of window is selected when the window is defined by the command:

POKE **TYPE**, *Type Number*

as in section 2.3.1 (A type 1 window was defined there). Invoking this command also results in subsequent windows, where the type is not specified, being of that type.

Type 0 windows are more complicated than other types in that they involve text. The number of lines of text allowed in a window is one less than the number of rows in that window, each having length two less than the number of columns in the window. The first line is different from any other in that it forms the heading of the window, and is placed centrally with respect to the window. All other lines are printed from the left hand edge of the window.
A BASIC subroutine is included in Appendix D, indicating how text is placed in memory. The routine converts text line by line into character codes, which are then poked into memory starting at address **TEXT**. If there is no text in the line or a line is incomplete then the value 0 is poked in to represent the end of line(EOL).
The heading(line 0) is printed on a darker background to make it stand out.

2.3.3 HIGHLIGHTING WITH A WINDOW

It is often useful (especially with type 0 windows) to highlight an area within a window that represents an option [See **IconEdit** for a typical example]. Each window is divided into a number of horizontal fields referenced by a number. The first (the heading in type 0 windows) by 0, the second by 1 and so on.

To highlight an area in a window:

POKE **FIELD**, *Field Number*

where the field number refers to the field in the window that was last defined, and then:

LET L = USR **HIGH**

Highlights are removed by repeating the sequence of commands.

2.3.4 REMOVING WINDOWS

The Toolkit would not be very useful if windows could only be placed on the screen. This would provide only 16 static windows. The Toolkit also has commands which allow the removal of windows from both the screen and memory. Each window has assigned to it a number, given to it when the window was defined. This forms an index to the windows and gives a means of referencing which window is to be deleted. To delete a window:

POKE **DEL**, *Window Number*

where the window number is in the range 0 to 16. Window numbers 1 to 16 delete the corresponding window, whereas if a value of 0 is assigned to it then the last defined window is deleted. To delete the window:

LET L = USR **REMOVE**

It is possible to delete any window even if it is partly or fully hidden by one or more other windows. If a window is deleted other than the last one defined then the numbering of the windows will change. All windows whose number was larger than that of the one deleted will have their number decreased by one.

2.3.5 USING THE MOUSE

The previous sections have dealt with the manipulation of windows. The benefit of the Toolkit is in the linking of these techniques with the mouse. The mouse can be used as an alternative to the keyboard to move a pointer about the screen, selecting options within windows by 'clicking' on that window. 'Clicking' is the process of moving onto a window and pressing one of the mouse buttons. How does the toolkit control this function?

The screen pointer can be initialised by:

LET L = USR **START**

A pointer is placed in the centre of the screen. To move the pointer around the screen:

LET L = USR **MOVE**

The mouse can now be used to move the pointer freely about the screen, continuing until one of the mouse buttons is pressed. At this point a value is assigned to the variable L. If L holds a value greater than zero, then L refers to the number of the window that was 'clicked'. If L is zero, then the pointer did not lie within the boundaries of a window when the button was pressed.

The Toolkit also provides a method of finding which button (of two in the case of the Kempston mouse) was pressed:

LET L = USR **BUT**

If L holds the value 1 then the right hand button was pressed, the value 2 then the left hand button.

The current screen co-ordinates of the pointer can be determined with:

```
LET X = PEEK XCRD
LET Y = PEEK YCRD
```

where XCRD and YCRD use the window definition co-ordinate system.

The pixel position of the pointer can be determined with:

```
LET X = PEEK XPCRD
LET Y = PEEK YPCRD
```

This does not give the same pixel co-ordinates as described in the Spectrum User Guide. All points are relative to the top left hand corner of the screen. The purpose being to allow reference on the 23rd and 24th line of the display.

Section 2.3.3 dealt with the methods of highlighting fields within a particular window. To determine which field of a particular window the pointer is in:

```
LET Y = PEEK FIELD
```

and to determine the X co-ordinate within the window:

```
LET X = PEEK FILDX
```

The commands introduced thus far provide an 'arrow' as the pointer. Toolkit provides a facility for changing the shape of the pointer, with up to four pointers being able to be held in memory at any one time. The pointers, in the same way as icons, are defined in the Icon/Pointer Editor **IconEdit** (Section 2.3.6).
To select a particular pointer:

```
POKE POINT, Pointer Number
```

The pointer will change when the next **START** or **MOVE** command is executed successfully.

The pointer can be removed from the screen with:

```
LET L = USR FINISH
```


It is not necessary to remove the pointer from the screen when a new window is inserted or a old one is deleted.

2.3.6 THE ICON/POINTER EDITOR

The previous sections have introduced the concept of icons and pointers. **IconEdit** provides a means of defining both icons and pointers for use within the Toolkit. The Toolkit supplies 8 icons and 4 pointers, these are not fixed and can be changed if so desired.
After successfully loading **IconEdit** , a menu will appear in the top left hand corner of the screen. The options are selected by moving the on-screen onto the menu and pressing a mouse button.

The Icon/Pointer Editor options are as follows:

EDIT ICON : Clicking this producse another window displaying 8 icons. Choose one of these by clicking it, and a magnified version of the icon will appear. Each individual pixel of the icon can be set or reset by pressing one of the mouse buttons while the pointer is over it. Any changes will also be reflected in the window containing the other icons.

EDIT POINTER: This option is similar to the edit icon option, but there are only four pointers. When chosen, two windows will appear. The first is the magnified version of the pointer, the second is known as the mask. The mask is placed on the screen before the pointer, and for every pixel turned on in the mask the corresponding pixel is turned on the screen is turned off. This results in a smooth scroll of the pointer on the screen. It should be borne in mind when editing a pointer, that the top left hand corner of it is used to decide whether it points to anything.

TEST POINTER: Again the four pointers are displayed, and the pointer that is chosen is that which is used when the mouse is moved. This option can be useful to indicate how well a mask fits a pointer.

FILE: Another menu will appear when selecting this option. This allows options to save the icons, the pointers or both to cassette. These can be loaded back into the editor by selecting the load option in this menu, or loaded into the Toolkit itself by typing the following in BASIC when the toolkit is in memory:

```
LOAD "" CODE
```

EXIT: Selecting this option takes you out of the editor, and returns to BASIC. The editor can be entered again by typing RUN.

To return to the main menu or any previous menus whilst in the Toolkit, just click that particular window.

The Icon/Pointer Editor has been written using the Toolkit, and shows only some of the features of what it can do. It occupies about 1K of memory, the vast majority of which is taken up to store text in the windows.

2.3.7 TOOLKIT ERRORS

Errors can occur whilst executing a Toolkit instruction, but some simple error detection is built into the Toolkit command structure.
Consider having defined only 4 windows and attempting to delete window 6. So, having input:

```
POKE DEL,6
LET L = USR REMOVE
```

L will hold the value 11 after the commands are executed. Appendix C lists all the errors that can occur whilst executing Toolkit commands. In general if L holds the value 0 after an instruction has been execute, then no errors will have occured. The only exception being **MOVE**, which normally returns a non-zero value. In the example above the non-zero value indicates an error, and Appendix C lists Error 11 as an attempt to delete a non-existant window. This provides a method or error-trapping.

The majority of errors occur when defining windows. The most fatal being Error 1, when there is a memory overflow and there is not enough memory to store the defined window. The Toolkit sets aside an area of memory to store the information underneath a window when it is defined. This error will only occur when a large number of sizeable windows are defined. This can be avoided by reducing the size of the windows, or changing the amount of memory allocated. However, a CLEAR can only be executed successfully when there are no windows defined, otherwise the program will be corrupted.

To allocate more memory, decide on the amount required (6000 bytes is usually sufficient), and subtract the value from 61660 to give MEM. To allocate more memory:

```
POKE DATA+1, INT (MEM/256)
POKE DATA, MEM - 256 * PEEK (DATA+1)
CLEAR MEM - 1
```

The last statement reserves more memory for the Toolkit.

Appendix C includes details of the Toolkit commands in which particular errors can occur, which should assist in error detection.

2.3.8 USING THE TOOLKIT IN YOUR OWN PROGRAMS

The Toolkit was designed for use in your own programs. The code sits at the top of memory, but does not overwrite the user defined graphics and does not create problems with the Sinclair Interface 1 and Microdrives. To make a copy of it for use in your own programs, load **Toolkit** then:

```
SAVE " Toolkit" LINE 9999
SAVE "Toolc" CODE 61660, 3708
```

In addition to the Toolkit code this will also save the Icons and Pointers that are held in memory at the time of saving. It is not essential to save the BASIC part since this is only present to assist in entering text for type 0 windows (see Appendix D).

To load the Toolkit into your own programs:

```
CLEAR 61659
LOAD " Toolc" CODE
```

The CLEAR ADDRESS may be different, as this depends on the amount of memory that you wish to reserve for the Toolkit. (See Section 2.3.7).

Now that you have loaded the Toolkit, what can you do with it? The choice is yours!!!

3.0 TECHNICAL SPECIFICATION

The Kempston mouse is port mapped, simple read instructions allowing the status of the mouse to be monitored at any instant:

```
X co-ordinate of mouse at 64479 (&FBDF)
Y co-ordinate of mouse at 65503 (&FFDF)
Status of trigger buttons at 64223 (&FADF)
```

The X and Y co-ordinates are returned as eight bit values corresponding to the relative movements of the mouse.
The trigger buttons are independent, the right hand button corresponding to bit D0 and the left hand button to bit D1 of port 64223. Both read zero on depression.

A special version of the Kempston mouse is available for use with the Disciple. The addresses for the Disciple variant of the Kempston mouse are as follows:

X co-ordinate of the mouse at 64503 (&FBF7)
Y co-ordinate of the mouse at 65527 (&FFF7)
Status of the trigger buttons at 64247 (&FAF7)

In all other respects the functioning of the mouse is exactly the same as above.

APPENDIX A

The following table gives a description of all the memory addresses that the Kempston Mouse Toolkit uses:

ADDRESS	NAME	POKE	PEEK	DESCRIPTION
61678	NOW	XXXX	0-16	No. of windows currently in use
61679	DEL	0-16	NNNN delete	Number of the window to
61680	TYPE	0-3	NNNN	Boxed, plain or empty window.
61681	XPOS	0-31	NNNN	X position of the window
61682	YPOS	0-23	NNNN	Y position of the window
61683	XSIZE	1-32	NNNN	X Size of the window
61684	YSIZE	1-24	NNNN	Y Size of the window
61685	COL	0-255	NNNN	Colour attributes of the window
61686	ICON	0-7	NNNN	Icon to be used for type 2
61687	POINT	0-3	nnnn	Pointer to be used on screen
61688	BUT	NNNN	1-2	Number of button pressed on mouse
61689	FIELD	0-Y	0-Y	Field highlighted or pointed to
61690	FILDX	NNNN	0-X	X Position within window of mouse
61691	XCRD	NNNN	0-31	X Position of the mouse
61692	YCRD	NNNN	0-23	Y Position of the mouse
61693	XPCRD	NNNN	0-255	X Pixel position of the mouse
61694	YPCRD	NNNN	0-191	Y Pixel position of the mouse
61695	DATA	CCCC	nnnn	Address space needed for toolkit
61697	TEXT	0-255	nnnn	Address of text for type 0

N.B.

Where NNNN appears in the table, it means that POKEing this address will have no lasting effect and PEEKing it will return a meaningless value. Where XXXX appears in the table, it means that this address should not be POKEd, since it would result in corrupting the program. The CCCC in the table refers to the address in the memory of where the toolkit work space starts. Sensible values start at round 26000 and go right up to about 55000. See section 2.3.7 for more details.

APPENDIX B

The following table gives an address and description of all the Kempston Mouse Toolkit instructions:

ADDRESS	NAME	DESCRIPTION
61660	START	Initialises the on screen pointer and places it in the middle of the screen
61663	MOVE	Allows the user to move the pointer about the screen. When one of the buttons is pressed it decides which (if any) window it is in and returns to BASIC.
61666	FINISH	Removes the on screen pointer from the screen.
61669	SETUP	Sets up a window and places it on the screen in the desired format
61672	HIGH	Highlight a specified field in the last defined window
61675	REMOVE	Removes a window from the screen

APPENDIX C

The following table gives a description of all the errors that can occur when a Kempston Mouse Toolkit instruction is performed:

CODE	NAME	DESCRIPTION
0	ALL	The instruction was executed successfully and no errors occurred
1	SETUP	There is not enough memory left to store this window
2	SETUP	This window can not be set up because 16 window have already been defined.
3	SETUP	The type specified for this window is invalid; it must be boxed (0), plain (1), empty (2) or icon (3).
4	SETUP	The YPOS parameter for this window is out of range; it must be in the range 0-31.
5	SETUP	The XSIZE parameter for this window is invalid; it can not have the value 0, or it must be 2 for icons.
6	SETUP	The XSIZE parameter for this window is out of range; XPOS+XSIZE must be in the range 1 to 32.
7	SETUP	The YPOS parameter for the window is out of range; 0 to 23.
8	SETUP	The YSIZE parameter for this window is invalid; it can not have the value 0, or it must be 2 for Icons.
9	SETUP	The YSIZE parameter for this window is out of range; YPOS+YSIZE must be in the range 1 to 24.
10	SETUP	The ICON parameter for this window is out of range; it must be in the range 0 to 7.
11	REMOVE HIGH	There is no window with this number (DEL) associated with it so it can not be deleted, or there is no window that can have a field highlighted.
12	HIGH	The FIELD parameter for this window is out ofrange; it must be in the range 0 to YSIZE-1.
253	START MOVE	The POINT parameter is out of range; it must be in the range 0 to 3.
254	START	The pointer has already been initialised.
255	MOVE FINISH	The pointer has not been initialised yet.

N.B.

The error code is the value returned by the USR call when a toolkit instruction is executed. The second column gives the toolkit instruction that the error is associated with.

APPENDIX D

The following program is supplied with the toolkit and aids the inputting of text for type 0 windows:

```
9900 READ Num, Width: LET Addr=61697
9910 FOR n=1 Num: READ a$
9920 FOR f=1 TO LEN a$
9930 POKE Addr,CODE a$ (f): LET Addr=Addr+1
9940 IF f=Width-1 THEN GO TO 9960
9950 NEXT f: POKE Addr,0
9960 LET Addr=Addr+1
9970 NEXT n: RETURN
9980 CLEAR 61659: LOAD "Toolc"CODE
```

To use it construct a BASIC data line as follows; The first two entries are the number of lines of text (one less than the height of the window) and the width of each line (two less than the width of the window). Then comes the text starting with the text for the heading. To put the text in memory use the RESTORE statement (pointing to the line containing the data) and call the routine using GO SUB 9900. An example would be:

```
100 DATA 4, 6, "MENU", "Edit", "File", "Exit"
110 RESTORE 100: GO SUB 9900
```

Any line of text that is too long will be truncated, and any line that is too short will be padded out. If you do not want text on a particular line then just use empty quotes ("").