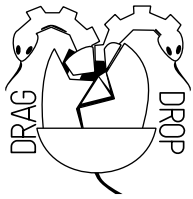
The Application Tutorial and Listings Book

for RISC OS computers

Chris Dewhurst

The Application Tutorial and Listings Book

for RISC OS Computers



DRAG'NDROP

www.dragdrop.co.uk

The Application Tutorial and Listings Book for RISC OS Computers

ISBN 978-1-80068-130-9
First published 2021 by
Independent Publishing Network (IPN)
https://bookisbn.org.uk

The author is indebted to the past and present writers of fine software applications for the RISC OS computer platform, without which this book would have been very difficult to produce.

Produced on RISC OS computers using Impression Style, Draw, Artworks, PostScript3 printer driver, and the Kyocera Ecosys M5521cdn multifunctional device.

Printed and bound in Great Britain by
Drag 'N Drop Publications,
Dingwall, Scotland.

www.dragdrop.co.uk dragdrop@dragdrop.co.uk
Typeset in 11/12pt Plato.

Acknowledgements

This book would not have been possible without the following people. Thanks to Tony Bartram at www.amcog.co.uk for help in testing the Envelope Editor (Chapter 8) which was written for the RDSP module (written by Tony). Thanks to Geoff McVeigh for help testing the Plotter and Notepad (Chapter 10). Thanks to the fiendly assistance of members of the forum at riscosopen.org.uk, not forgetting the work done by ROOL on RISC OS itself. Finally thanks to Sybil Harris at www.sybilharris.com for once again providing the amazing front cover artwork.

Contents

1 Introduction

This book is about writing multi-tasking applications, or 'apps' for short, for the RISC OS desktop, or GUI (Graphical User Interface). It is written in similar style to the *Drag 'N Drop* magazine, which is all about making computer programming on RISC OS a hands-on, fun and accessible experience.

Traditionally, there has been a shortage of RISC OS reference material. What exists is excellent for the experienced technician but impenetrable for the beginner. Incidentally, when we say 'beginner' we mean you are familiar with the RISC OS desktop from a user point of view (iconbar, running programs, using the three-button mouse, opening directories and so forth) and have reasonable knowledge of BBC Basic (including elementary graphics using PLOT and DRAW) but haven't necessarily done any application coding.

Previous books on writing for your desktop tended to concentrate on building a single application, adding more features as chapters went on. Listings tended to be long winded, sometimes relying on libraries of code only provided on a floppy disc – a medium that is now digitally redundant.

This book is different, however. It consists of lots of small (under 6K) demonstration listings, all of them printed in the pages of this book. None of them need vast libraries of routines, except for short procedures introduced as we go along and all listed in the appendices. Some programs are taken from the pages of *Drag 'N Drop*, many are complete applications in themselves to perform some useful task like converting a file, which is easy to do with the reknowned 'drag and drop' operation of RISC OS.

Programs

Desktop apps can be written in virtually any language available on RISC OS – Basic, Assembler, C, etc. We use BBC Basic in this book because BBC Basic has come free with every RISC OS distribution, built into the ROM chips in the Acorn 'A' machines of the 1980s right up to the ROM images on SD cards for Raspberry Pi's in the 2020s.

You're assumed to have some proficiency with BBC Basic. Many people know the language from growing up with a BBC Micro but if you are completely new then we refer you to at least the first few chapters of the *BBC Basic Reference Manual*. We recap on general fiddling around at binary level in Chapter 3. (Bitwise manipulation, although discussed in older generic books

2 The Desktop

We are familiar with the RISC OS desktop from everyday use – double clicking icons, typing documents and spreadsheets, dragging and dropping etc. In this chapter we take a slightly deeper look with short demo programs using simple BBC Basic graphics to illustrate aspects of the system to help us get a feel for what's in store.

The INFO program below is a very simple application and it introduces our first three 'Wimp' calls, WIMP being the traditional computer programming acronymn for Windows Icons Menus and Pointers. As you move the pointer around the desktop, the text at the top left displays the pointer's coordinates in *graphics units* (also known as *OS units*), the *handle* of the window and the icon underneath the pointer. Press Escape to stop.


```
REM VERY SIMPLE APP
SYS "Wimp_Initialise",200,&4B534154,"Info"
DIM B 256
PROCGET
PROCUPDATE
REPEAT
SYS "Wimp_Poll",,B
PROCGET
IF X<>OX OR Y<>OY PROCUPDATE
UNTIL INKEY-113
SYS "Wimp_CloseDown"
END
:
DEF PROCGET
SYS "Wimp_GetPointerInfo",,B
X = !B : Y = B!4
W = B!12 : I = B!16
ENDPROC
:
DEF PROCUPDATE
VDU 5 : GCOL 131 : GCOL 15
PRINT TAB(40,0) STRING$(40,CHR$127) "(" ; X ; "," ; Y ; ")" ;
PRINT "WINDOW &";~W;" ICON ";I
OX = X : OY = Y
ENDPROC
```

INFO size=429 bytes, CRC=38603

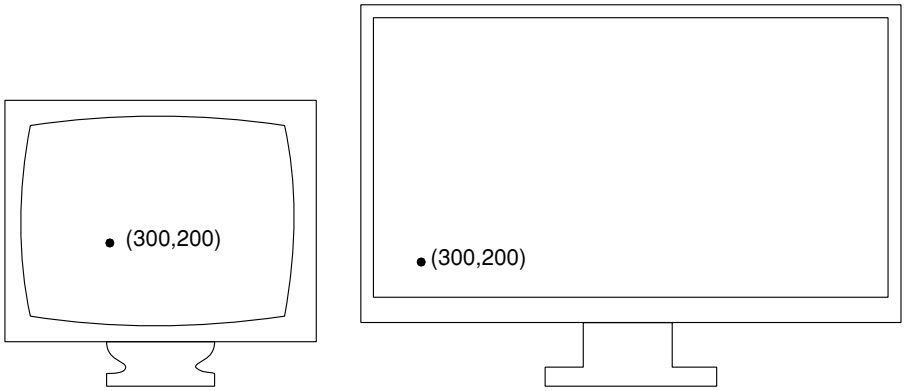
A *handle* is the computer's way of identifying the window or icon. When interacting with the system we generally always have to give the *handle* of

thing thing we are talking about – *window handle*, *icon handle*, *font handle* and *file handle* frequently crop up in RISC OS programming.

Size, Resolution, and Graphics Units

Move your mouse pointer to the bottom left of the screen so the coordinate readings are (0,0). Now move the pointer diagonally, up and to the right. How far can you go? The top right coordinates of my desktop are (3838,2158) so the *size* of my desktop display is 3838 × 2158 graphics units. Load up Paint and click Menu on the iconbar icon and choose Snapshot. Ensure **No delay (instant effect)** is selected, and choose **Grab whole screen**, then click OK. Drag the sprite in the 'save as' box back onto Paint's icon bar icon. Click Menu > Misc > Sprite. What is the width and height? Mine is 1920 x 1080.

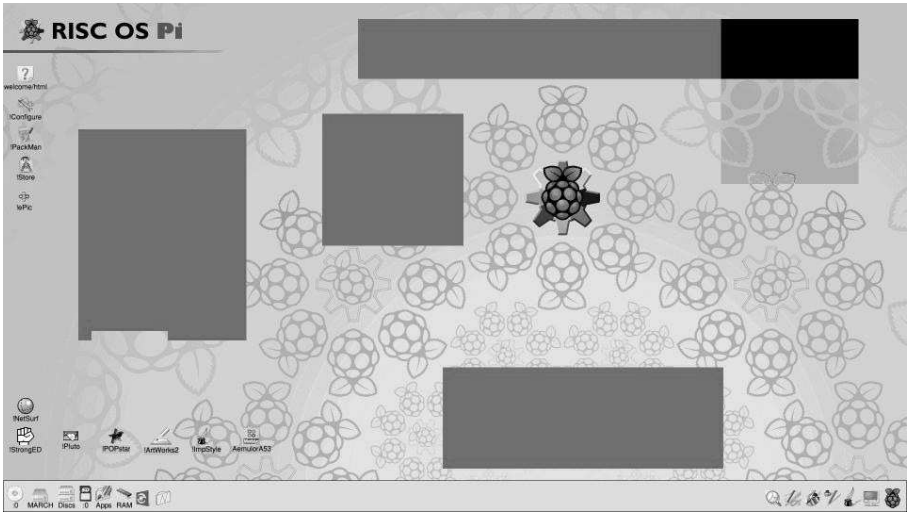
The desktop size is different its *resolution* which is the number of pixels. On modern RISC OS displays there are 2 graphics units per pixel, so when we talk of coordinates (300,500) on the screen we are actually referring to the 150th pixel from the left and the 250th pixel from the bottom of the screen.



In the past we may have referred to a screen *size* of 1280 by 1024 graphics units (on the BBC Micro or Archimedes say) and depending on the screen mode this meant a resolution of 640 x 480 pixels. Graphics coordinates (300,200) would be about a quarter of the way across the screen and a quarter of the way up on an old 640 × 480 display but on a display of 3838 × 2158 graphics units only a tenth of the way across and up. Don't fret if you don"t understand this, just be aware that when we come to put stuff on the screen the appearance and position can take you by surprise.

Every application must start with a line similar to line 2 of INFO. It uses a

overwriting. It draws a rectangle with one corner at (XC,YC) with width (X-XC) and (Y-YC).



Similar to INFO, if the new pointer coordinates are different from the old ones (either X<>OX OR Y<>OY) the PROCUPDATE is called with GCOL plotting option 3 for EORing the colour on the screen.

When you have had enough, press Escape and type F12 followed by Return to clear up the mess on the desktop.

```
SYS"Wimp_Initialise",200,&4B534154,"SHAPES"
DIM B 2000
PROCGET
XC=X:YC=Y
OX=X:OY=Y
REPEAT
SYS"Wimp_Poll",,B
PROCGET
IF BT PROCCLICK
IF X<>OX OR Y<>OY PROCUPDATE
UNTIL INKEY-113
SYS"Wimp_CloseDown"
END
:
DEF PROCGET
SYS"Wimp_GetPointerInfo",,B
X=!B:Y=B!4:BT=B!8
ENDPROC
:
DEF PROCUPDATE
```

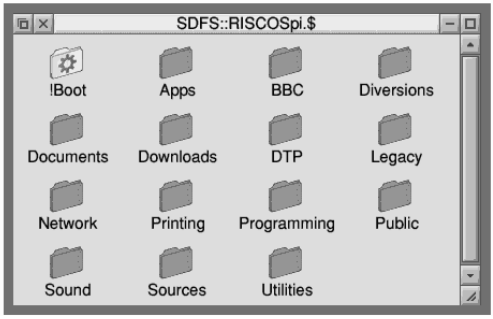
```
PROCPLOT(3)
RECTANGLE FILL XC, YC, OX-XC, OY-YC
OX = X : OY = Y
ENDPROC
:
DEF PROCCLICK
IF (BT AND 1)=1 PROCPLOT(3) ELSE PROCPLOT(0)
XC = X : YC = Y
ENDPROC
:
DEF PROCPLOT(K)
GCOL K,3
RECTANGLE FILL XC, YC, X-XC, Y-YC
ENDPROC
```

RECPAINT size=526 bytes, CRC=11659

Window Edger

This program draws a thick red line around the window under the pointer when the Ctrl+Alt keys are pressed. It gets the handle of the window under the pointer as in the INFO program, then uses a new call SYS call, "Wimp_GetWindowInfo". It takes as input the window handle at B. It returns the left edge X coordinate in B+4, bottom edge Y in B+8, right edge X in B+12 and top edge Y in B+16. These are stored in LX, BY, RX and TY respectively in line 15. The loop in lines 16-18 draws rectangles of increasing width and depth to give the thick line effect.

As before press Escape to stop and F12 followed by Return if you want to clean up any garbage.




```
SYS "Wimp_Initialise",200,&4B534154,"Window Rectangle"
DIM B 2000
REPEAT
SYS "Wimp_Poll",,B
```

```
SYS"Wimp_Poll",,B
IF INKEY-6 AND INKEY-5 PROCACTION
UNTIL INKEY-113
SYS"Wimp_CloseDown"
END
:
DEF PROCACTION
SYS"Wimp_GetPointerInfo",,B
W=B!12 :REM WINDOW UNDER POINTER
I=B!16 :REM ICON
B!8=2 :REM MENU BUTTON
SYS"Wimp_SendMessage",6,B,W,I
ENDPROC
```

KEYMENU size=341 bytes, CRC=20335

As before, Wimp_GetPointerInfo is used to get the handle of the window and icon under the pointer. A new call, SYS "Wimp_SendMessage" is used. We'll encounter messages later on in the book and you don't need to understand anything about them this stage, just that register R0 contains 6 (message number for "mouse click"), R1 the memory Block and registers R2 and R3 the window handle W and icon handle I respectively. This is the one call, by the way, which uses register R0 so no double commas !

Pulsating Pointer

This is a very short program which adjusts the colour of the mouse pointer by level L after a wait W (2 centiseconds), giving the illusion of a throbbing pointer. Press Escape to stop the effect.

The Basic statements MOUSE COLOUR 1 and MOUSE COLOUR 2 control the colours of the pointer outline and inside of the pointer. The colours are expressed in quantities of the three component colours red, green, and blue from 0 to 255. PROCCHANGE applies the level of colour to all three colours so the throbbing is in white through grey to black, however if you wanted a blue/pink pointer you would do MOUSE COLOUR 1, L, 0, L and MOUSE COLOUR 2, 255-L, 0, 255-L in line 20.

```
SYS"Wimp_Initialise",200,&4B534154,"Mouse colours"
DIM B 256
TI=TIME
```

```
W=2
L=1
REPEAT
SYS"Wimp_Poll",,B
IF TIME>TI+W PROCCHANGE
UNTIL INKEY-113
MOUSE COLOUR 1,0,255,255
MOUSE COLOUR 2,0,0,255
SYS"Wimp_CloseDown"
END
:
DEF PROCCHANGE
TI=TIME
L=(L+16) AND 255
MOUSE COLOUR 1,L,L,L
MOUSE COLOUR 2,255-L,255-L,255-L
ENDPROC
```

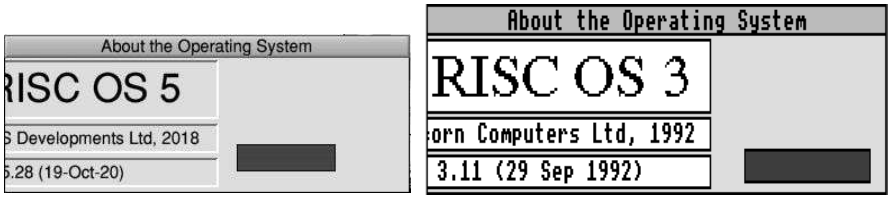
PULSE size=321 bytes, CRC=16131

Autoscroller

If you're feeling lazy and can't be bothered to press the scroll arrows beside windows, this program is for you. It uses a slightly extended version of the SYS "Wimp_Poll" call. It has TO E added to the end. The TO instructs the system to return information. E is the Event Number. We'll meet events later.

PROCSCROLL gets the pointer coordinates into X and Y and additionally reads the scroll offsets into XS and YS from B+20 and B+24 of the block after "Wimp_GetWindowState" is called. Scroll offsets are the horizontal and vertical bars you see at the bottom and right of some windows allowing you to pan over a document occupying a larger area than the window. If the pointer coordinates are within a certain distance of the edges of the window (16 graphics units in this case) then the scroll offsets are adjusted and Message Number 2 ("open window") is sent to the application.

Use Autoscroller to find the secret red button in the 'About this Operating System' window. Does it do anything if you click it? Try it and see!




```
SYS"Wimp_Initialise",200,&4B534154,"AUTO SCROLL"
```

3 Words, Bytes 'N Bits

This chapter is a mainly a recap on memory – your computer's memory that is. Computer programming is largely a reading-in-silence affair, as opposed to talking out loud, so we give some tips at pronouncing computer code. If you don't have anyone to talk to, it's still useful to improve your elocution. We cover some important theory which lays the foundation for our programming later on. It may seem tedious and if you really feel you are an expert on all this you can skip to the next chapter.

You will already know that the computer stores its numbers in bytes and there are 8 bits to a byte, a byte holding a number from 0 to 255. (Or 0 to 127 with a sign bit but we don't concern ourselves with that here.)

We saw in Chapter 2 that on RISC OS (and modern computers so far) four bytes are grouped into a *word*. Our friend &4B534154 is a word, sometimes referred to as a *four-byte integer* or *32-bit integer* in older books on BBC Basic. It consists of four bytes &4B, &53, &41 and &54. There are 8 bits in a byte so 8 × 4 = 32 bits in a word. The ampersand (&) indicates 'hexadecimal'. The ampersand is not a computing standard and lots of people use 'x0' (lower case X and zero) to mean hexadecimal, or hash (#), dollar ($) or even all four (&#x0). BBC Basic only understands the ampersand, though.

We also saw in Chapter 2 that &4B534154 is the four Ascii codes (American Standard Code for Information Interchange - say "AHSS - key") for "TASK" spelt backwards. It's a 'magic word' needed to activate applications, the reasons for which are best left to the history books.

! (Pling)

You can put (poke) a word into the computer's memory and read it back (peek) with ! (pling, bang, exclamation mark but known as pling on RISC OS). It's called an operator. You must have reserved a block of memory with DIM to use the operator. Type Ctrl+F12 to open a task window, type BASIC and press Return.

```
ARM BBC BASIC V (C) Acorn 1989

Starting with 651516 bytes free

>
```

Type the following lines at BBC Basic's > prompt. You say "Dim Bee one

hundred, pling Bee equals hex four bee, five three, four one, five four. Print tilde pling Bee".

```
DIM B 100
!B=&4B534154
PRINT ~!B
  4B534154
```

The ~ (tilde) tells the computer to “print in hexadecimal”.

? (Query)

The query or question mark operator lets you poke or peek bytes. Assuming you have typed in the above, try

```
PRINT ~?B,~B?1,~B?2,~B?3
        54        41        53        4B
```

It's the same as &4B534154 but in reverse. The &54 is called the *low* byte and appears at the end of &4B534154 but is the *first* byte in memory. The &4B is the *high* byte, at the beginning of &4B534154 but the last byte in memory. This ‘arse-about-faceness’ dates from a time when computers were slower and had less memory. If you wanted to store the number 100 (hexadecimal &64) in memory, that only needs one byte. Expressed as a word that's &00000064. The &64 might as well be stored first in memory since the other three bytes are zero.

$ (String)

The dollar or string operator in BBC Basic is used to peek or poke strings. Type the following at the prompt (say "string Bee equals open quotes risk space oh ess" close quotes, Print string Bee"):

```
$B="RISC OS"
PRINT $B
RISC OS
```

Now type

```
FOR I=0 TO 7:PRINT I?B:NEXT
        82
        73
        83
        67
```

4 Windows

In Chapter 2 we took out time messing around on the RISC OS desktop with some primitive multi-tasking programs, meeting some SYStem calls and (if you chose to read Chapter 3) surviving some in-depth discussion about bits and bytes. This gives us a good foundation for the road we are now about to set out on, starting with putting a window on the screen.

Windows don't suddenly appear the minute you tell RISC OS about them. First of all there is a setting up stage (telling the computer characteristics of a window like colour and size). One or more windows can be established at the start, in the application's initialisation section, or even 'on the fly' in response to actions the user is carrying out. Then the computer has to be instructed to open windows. Depending on what the program does not all windows will appear at once when the app launches. Usually their arrival on the scene is in response to a click or a menu selection.

Program WIND1 listed below puts a small window onto the screen using a procedure PROCWINDOW and a function FNKWINDOW short for "make window". FNMKWINDOW is the same in every listing and the format of the DATA statements in PROCWINDOW is the same, although individual items of data will vary according to the window's exact appearance.

Lines 2 makes the initialisation call, line 3 DIMensions two areas of memory, the Block and another one called T which is to store the window title data in. I have allocated 256 bytes for both, which is fine for most purposes but you may need to increase the size of T later on. Q is a flag which will be TRUE when the application is to quit either as a result of the user pressing Escape or a message to shut itself down – more on this later.


```
REM Window
SYS "Wimp_Initialise",200,&4B534154,"A Window"
DIM B 256, T 256
```

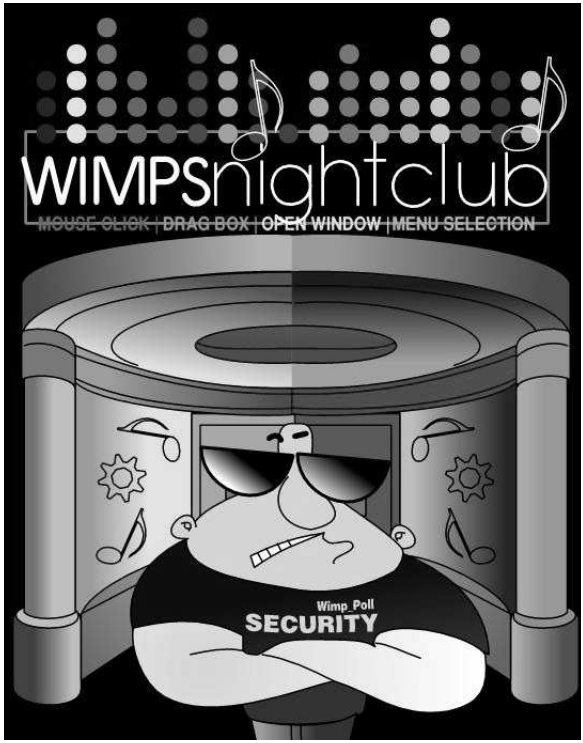
```
Q=FALSE
PROCWINDOWS
!B=W1
SYS"Wimp_GetWindowState",,B
SYS"Wimp_OpenWindow",,B
REPEAT
SYS"Wimp_Poll",,B TO E
IF E=2 SYS"Wimp_OpenWindow",,B
IF E=3 SYS"Wimp_CloseWindow",,B:Q=TRUE
UNTIL Q OR INKEY-113
SYS"Wimp_CloseDown"
END
:
DEF PROCWINDOWS
RESTORE+1
DATA "***Press Escape***":REM Window Title
DATA 100,150,400,300:REM Bottom left coords plus width and height
DATA X,Y,X+W,Y+H,0,0,-1
DATA &FF001012:REM window flags
DATA &03000207,&000C0103:REM window colours
DATA 0,0,W,H:REM Work area
DATA &00000101 Title Flags
DATA 0,0,0,T,0,0,0
W1=FNMKWINDOW:REM WINDOW
ENDPROC
:
DEF FNMKWINDOW
READ $T,X,Y,W,H
FOR I=0 TO 84 STEP 4
READ A$
I!B=EVAL A$
NEXT
T+=LEN $T+1
SYS"Wimp_CreateWindow",,B TO X
=X
```

WIND1 size=773 bytes, CRC=62317

A call to PROCWINDOWS is made in line 5. We'll come back to lines 6-16 shortly. In PROCWINDOW (singular) Basic's data pointer is RESTOREd to the next line with the command RESTORE+1. Lines 19-26 are DATA statements giving information about window characteristics and line 27 gets a handle for the window into W1 by calling FNMKWINDOW.

In this book we use eight DATA statements for every window definition, they look more or less like this:

DATA Window Title : REM must not contain commas
DATA 300,350, 500,300 : REM bottom left X Y and width height in graphics units



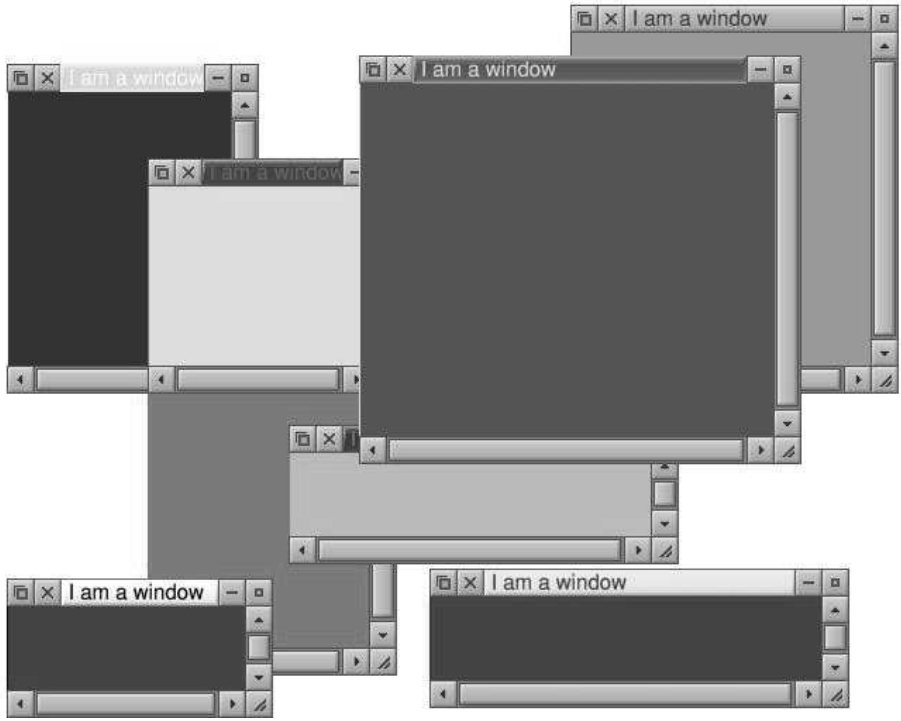
A complete list of events is given in the appendices, here we only need to know about events number 2 and 3. Event Number 2, open window, is rather strange – why would we want to open a window if it's already opened by the call in line 8? When you drag a window around the screen the window is actually closed at the old position and reopened at the new position. This is all done automatically by RISC OS but try removing line 11 and see what happens.

Event Number 3, close window, is delivered to our application when the user clicks on the close icon. So we respond by setting the Q flag to TRUE so that the REPEAT..UNTIL in line 13 will activate. The application quits when either the window is closed or the Escape key is pressed.

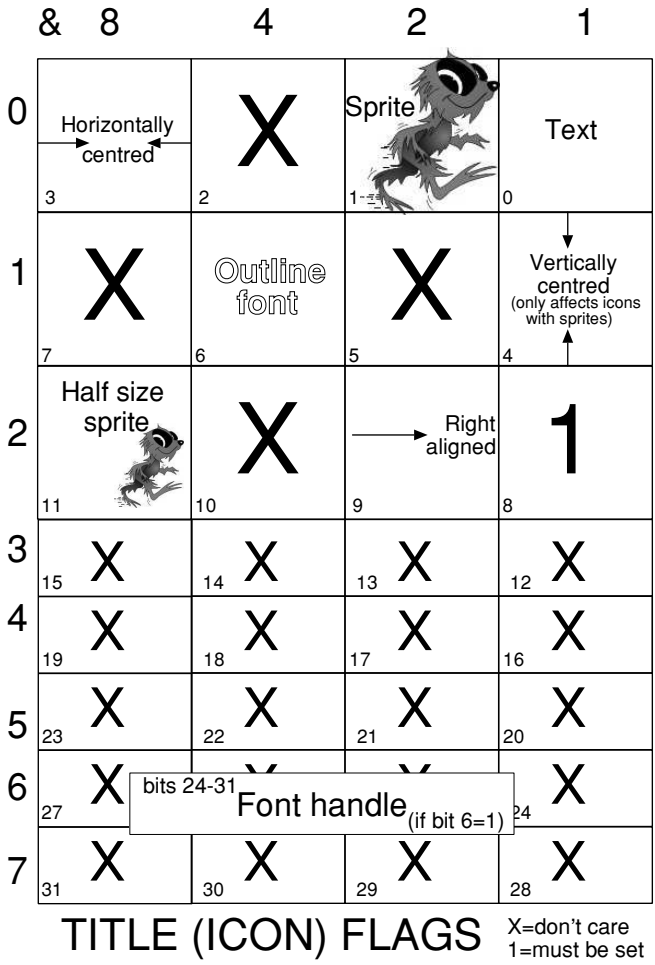
Incidentally, the statement UNTIL Q is the same as UNTIL Q=TRUE and the same principal applies for any variable that is non-zero. TRUE has value of -1 in the computer's memory and you can also write IF Q THEN... as a shorthand for IF Q=1 THEN...

The next program puts 10 windows on the screen of random colours and

random dimensions and positions. You will need to add FNMKWINDOW from the last program to complete the listing. Press Escape or close one of the windows to quit.




```
SYS"Wimp_Initialise",200,&4B534154,"Random windows"
DIM B 256, T 256, W(9)
Q=FALSE
PROCWINDOWS
FOR J=0 TO 9
!B=W(J)
SYS"Wimp_GetWindowState",,B
SYS"Wimp_OpenWindow",,B
NEXT
REPEAT
SYS"Wimp_Poll",,B TO E
IF E=2 SYS"Wimp_OpenWindow",,B
IF E=3 SYS"Wimp_CloseWindow",,B:Q=TRUE
UNTIL Q OR INKEY-113
SYS"Wimp_CloseDown"
END
:
DEF PROCWINDOWS
FOR J=0 TO 9
```



Window Colours

There are usually 16 colours available for windows and icons, numbered 0 to 15. They can be altered and you can get more colours which we discuss in a

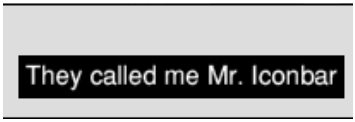
Icon Bar Icons – Text

The next program puts text onto the icon bar

```
SYS"Wimp_Initialise",200,&4B534154,"Iconbar Text"
DIM B 256, T 256, U 256, V 256
PROCMKICON(-1,0,0,160,50,&70000121,"They called me Mr. Iconbar",1)
REPEAT
SYS"Wimp_Poll",,B
UNTIL INKEY-113
SYS"Wimp_CloseDown"
END
:
```

Insert PROCMKICON

ICONBART size=404 bytes, CRC=36429



Style

In the previous program the icon's width and height measured 380 × 50 graphics units. Until you've programmed a few applications, becoming proficient with designing icons is largely guesswork. I started with 300 × 50 and found the text was cut off because I hadn't allocated enough width. Don't be afraid to experiment. Press Escape, fiddle with the numbers and re-run the program until you are happy.

There are guidelines as to how much space you should leave between icons, distance from the edges of the window, colours and so on, prescribed in the RISC OS *Style Guide*. It isn't the Fashion Police for your applications but is worth purchasing the book and observing the guidelines whererever you can so that desktop software written by different authors all have the same feel.

This time Icon Flags are &70000121. Bit one is set to indicate the icon contains text. Bit five is set for a filled background, and bits 28-31 are 7 (the leftmost nybble in &700000121) which is the Wimp colour for black. Try &B70000121 for black text on red. The left most nybble is B the hex code for red, the next nybble is 7, the code for black. Or try &B7000125 to get a border round the icon. Bit two is set for a border and the border's colour is the same as the text colour.

Icon Bar Icons – Text and Sprite

Programming an icons to display both text and sprite is rather tricky because it's difficult to work out the effects you want, owing to so much information being packed into a small space. What happens is that the justification bits (that is bits 3, 4 and 9 in the Icon Flags) are used to control where the sprite is in relation to text. The text is given in the A$ parameter of PROCMKICON as before but this time the sprite's name is poked into memory at V, prefixed with an 'S'. The last paramter in PROCMKICON is set to V and not 1 as before. As if this wasn't complicated, you have to ensure the bottom left corner of the icon, plus it's dimensions, are right so that everything works as expected.

The following program demonstrates. There is a built-in icon called SWITCHER so this is prefixed with S (to give “SSWITCHER”) and put in memory at V in line 3. The Icon Flags have bits 0 (text), 1 (sprite), 2 (border) and 3 (horizontal centre) set, plus the text colour of 7 (black) is specified in bits 24-27. The height of the icon is 120 to ensure the sprite can be accommodated. This gives the effect of the sprite above the text which is a tradition (look at the left of your icon bar, your CD rom drive, SDFS drive and so on all have the text ":0" below their icon).

```
SYS"Wimp_Initialise",200,&4B534154,"Iconbar Text and Sprite"
DIM B 256, T 256, U 256, V 256
$V="SSWITCHER"
PROCMKICON(-1,0,-20,380,120,&0700010F,"They called me Mr. Iconbar",V)
REPEAT
SYS"Wimp_Poll",,B
UNTIL INKEY-113
SYS"Wimp_CloseDown"
END
:
  Insert PROCMKICON
  ICONBARTS size=436 bytes, CRC=52561
```

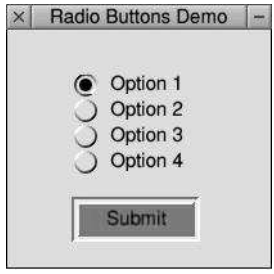
The text is on a white background, even though we haven't set the background colour bit. This always happens with text-plus-sprite icons, the text is 'highlighted' in the colour specified in bits 24-31 of the Icon Flags, which is zero, the Wimp colour for white. To work around this effect, set bits 24-31 to 1, which is the same shade of grey as the icon bar. Try &1700010B for a more 'traditional' looking icon bar icon

Validation Strings – F parameter

Colours for the outline font are specified in the validation string as F followed by two hexadecimal digits, the background colour (0-&F) and foreground colour (0-&F). "F1B" for example would produce red text on grey (85%) background. If your window work area was black you would use "F7B" to ensure the text blended in nicely.

Radio Buttons

Sometimes there is a need to select one (and only one) option from a group of items. Choosing one deselects the others. These are traditionally known as radio buttons, after an old fashioned radio set where you pressed one button to choose a preset station and it pushed out all the others.



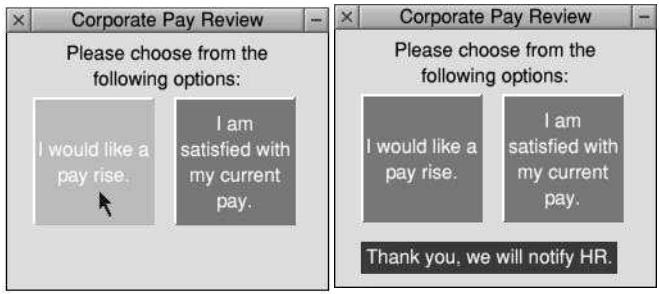

```
SYS "Wimp_Initialise",200,&4B534154,"Radio Buttons Demo"
DIM B 256, T 256, U 256, V 256
Q=FALSE
PROCWINDOWS : PROCICONS
!B=W1
SYS "Wimp_GetWindowState",,B
SYS "Wimp_OpenWindow",,B
REPEAT
SYS "Wimp_Poll",,B TO E
IF E=2 SYS "Wimp_OpenWindow",,B
IF E=3 SYS "Wimp_CloseWindow",,B
IF E=17 OR E=18 Q=(B!16=0)
UNTIL Q OR INKEY-113
SYS "Wimp_CloseDown"
END
:
DEF PROCWINDOWS
RESTORE +1
DATA "Radio Buttons Demo",500,500,410,370
DATA X,Y,X+W,Y+H,0,0,-1
```

Changing Icon Flags

Sometimes an application needs to change the appearance of an icon, other than its text, after it has been set up in PROCMKICONS. We now introduce a standard procedure PROCWRICONF (for WRite ICON Flags) using the system call SYS "Wimp_SetIconState". The Block contains four works: B holds the window handle, B+4 the icon handle, B+8 the EOR word and B+12 the clear word. If you don't understand what's meant by EOR words and clear words, go back and read the relevant section Chapter 3.

We said in the previous section that PROCWRICONT puts zero in the Block B+8 and B+12. As you may have realised, all that achieves is setting the clear word and EOR word to zero but it saves passing excessive parameters when all we want to do is change the icon's textual content.

The following program demonstrates how we can dynamically alter bit 22 (the 'icon greyed out bit') in an amusing way. PROCICONS set up the icons as usual. Icon zero is a label. Icons one and two have Icon Button Type 1 which causes the system to inform the program via a click event (event number 6) that the pointer is on the icon – even though the user hasn't actually clicked a mouse button. PROCCLICK examines the icon and if it's icon one, the most desirable option, it's immediately greyed out. The clear and EOR words are B22, a constant set to 1<<22 or &400000 in line 46.



If the pointer leaves the window, bit 22 of icon one's flags are reset (EOR word 0, clear word B22). The pointer leaving the window is Wimp event number four, detected in line 10.

So you have no choice but to try the least desirable option (!) and PROCUPDATE is called which flashes up a suitable 'corporatey' message. This is just icon three, set up in disguise (a label icon with grey text on grey background) being made visible with a call to PROCWRICONF. The clear word is –1 (i.e. zeroising all 32 flags) and EOR word of &80000121 (bits 28-31 dark

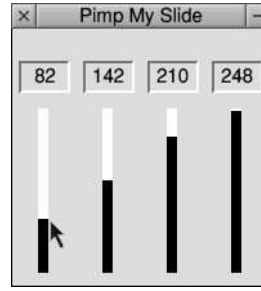
```
T+=□LEN□$T+1
SYS□"Wimp_CreateWindow",,B□TO□X
=X
:
DEF□PROCMKICON(H,X,Y,W,D,F,A$,V)
$U=A$□:□RESTORE□+1
DATA□H,X,Y,X+W,Y+D,F,□U,V,LEN□A$+1
FOR□I=0□TO□32□STEP□4
READ□B$□:□I!B□=□EVAL□B$
NEXT□:□U+=□LEN□A$+1
SYS□"Wimp_CreateIcon",,B□TO□X
ENDPROC
:
```

Insert FNMKWINDOW, PROCMKICON
SQUARED size=1738 bytes, CRC=55710

The PAPER sprite takes up a lot of memory and there is a more efficient way of drawing grids in windows which we will come to in the chapter on user graphics.

Slider Icons Take 2

The next program demonstrates an easier method of adding slider icons to your windows by employing user sprites.

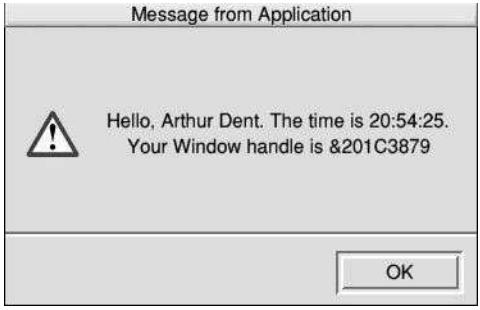



```
SYS□"Wimp_Initialise",200,&4B534154,"User□Sprite□Slider□Icons"
DIM□B□256,□T□256,□U□256,□V□256
N=3□:□REM□no□dials□-1
DIM□IV(N),IL(N),IU(N)
FOR□I=0□TO□N
IV(I)=RND(255)□:□IL(I)=0□:□IU(I)□=□255
NEXT
Q=FALSE
PROCSPRITES□:□PROCWINDOWS□:□PROCICONS
!B=W1
SYS□"Wimp_GetWindowState",,B
SYS□"Wimp_OpenWindow",,B
REPEAT
```

```
A$="Hello,□"+A$+".□The□time□is□"+RIGHT$(TIME$,8)+".□Your□Window□handle□is
      &"+STR$~W1
PROCREPORT(A$)
ENDPROC
:
DEF□PROCWINDOWS
RESTORE□+1
DATA□Input□text,200,200,600,200
DATA□X,Y,X+W,Y+H,0,0,-1,&86001012,□&01000207,&000C0103
DATA□0,0,W,H,&109,0,1,0,T,0,0,0
W1=FNMKWINDOW
ENDPROC
:
DEF□PROCICONS
$V="R6,13"
PROCMKICON(W1,20,100,400,50,&8000012D,"Please□type□your□name",0)
PROCMKICON(W1,20,50,400,50,&0700F12D,STRING$(30,CHR$13),0)
PROCMKICON(W1,450,50,100,100,&A700412D,"OK",V)
ENDPROC
:
```

Insert FNMKWIN, PROCMKICON, PROCRDICON, PROCREPORT, PROCERROR

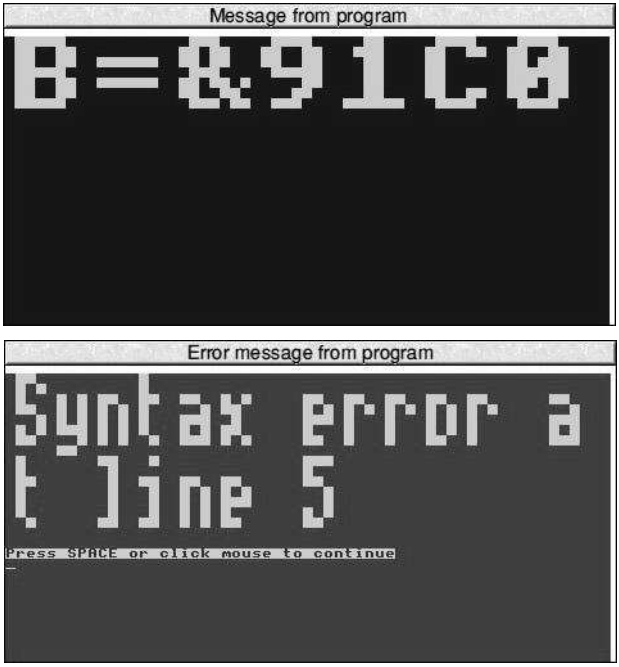
REPORT size=1490 bytes, CRC=26611



Text Windows

RISC OS can be made to display a text window in which normal VDU 4 characters can be displayed. You do this with SYS "Wimp_CommandWindow", taking no parameters or one. With no parameters, a text window is defined but not cleared. SYS "Wimp_CommandWindow",B will default to a white screen with black text with the text (control character terminated) at B displayed as the window title. In both cases, no other desktop operations can be carried out until you click the mouse or press a key. Also, if you have a high resolution monitor the

characters in the window are tiny! The next program demonstrates by blowing up the text using PROCBIG(A$,X) where A$ is the text and X=8 or 4 for full or half size characters, providing large-print alternatives to the previous PROCREPORT and PROCERROR.




```
ON ERROR PROCERROR
SYS "Wimp_Initialise",200,&4B534154,"Reporting errors"
DIM B 256
PROCREPORT("B=&"+STR$~B)
...deliberate error...
SYS "Wimp_CloseDown"
END
:
DEF PROCREPORT(A$)
$B="Message from program"
SYS "Wimp_CommandWindow",B
COLOUR 176 : COLOUR 63 : CLS
PROCBIG(A$,8) : A=GET
ENDPROC
:
DEF PROCBIG(A$,X)
COLOUR 255:COLOUR 0
FOR J=1 TO LEN A$
?B=ASC MID$(A$,J,1)
SYS "OS_Word",10,B
```

7 Dragging, Dropping and Working with Files

In Chapter 5 we learned about icons and how to make an application react to mouse clicks on icons – windows opening in response to 'action' buttons, dragging graphical bars (slider icons) and so on. One of the features of RISC OS that makes it so powerful and easy to use compared to other systems is the ability to drag an icon, or more generally an *object*, onto another icon to produce some result. For example, a file icon from a filer display that you pick up and deposit – drag and drop – on to the icon bar.

As users of RISC OS we take this for granted. From a programmer's point of view, however, it's a complicated topic which in this book we can only skim the surface of. This chapter is an primer giving you a good enough understanding to write small but very useful and fully functional applications, and putting you in good stead to have a crack at the Messages section in *The Programmer's Reference Manual*.

Drag–drop

Without trying to overload you with too much detail at this stage, whenever your app might need to know something's been dragged to it the system sends Event Number 18 to the polling loop and *Message* Number 3 plus a whole load of other information about the object in question. Again, *Messages* are a complex topic and for our purposes all we need to know is the word at B+16 contains 3.

The word at B+40 holds the file type, &00000FFB or just &FFB denoting a Basic file, for instance. If it was a directory B+40 is &1000. The filename starts at B+44 and this is the full *pathname* of the file, for instance **SDFS::RISCOSpi.$. Documents.UserGuide.StarComms**. The pathname is zero-terminated (if you don't understand this go back to Chapter 3).

The next program installs a Pool Sprite icon called SWITCHER onto the icon bar. There are a few lines REMed out which will be removed in due course to demonstrate different aspects. The window and icons are set up as usual, two icons for text and a third for a file icon which are padded out with CRs.

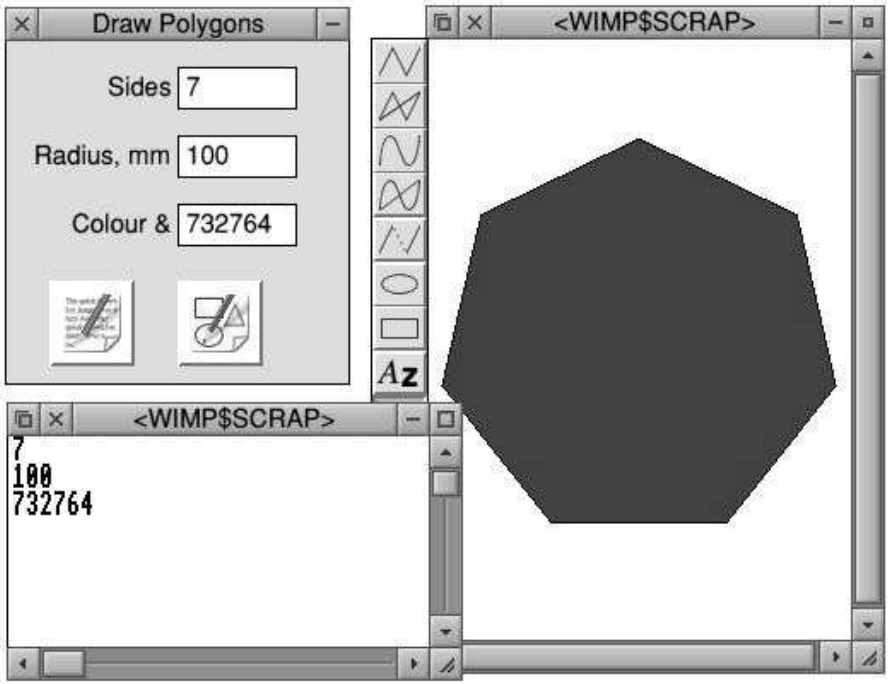
!BOOT.Resources.ScrapDirs.ScrapDir. In filing operations, such as our SYS call to retrieve the size of the file, instead of typing a long-winded SYS "OS_File",5,"!Boot.Resources.ScrapDirs.ScrapDir.SCRAPFILE" you just write SYS "OS_File",5,"<WIMP$SCRAP>". The WIMP$SCRAP (or Wimp$Scrap or wimp$scrap, the case doesn't matter) goes between the angular brackets.

RISC OS prior to 3.5 doesn't know about WIMP$SCRAP so it is necessary to set it up with a one-line obey file:

```
SET WIMP$SCRAP <OBEY$DIR>.SCRAPFILE
```

Save it as **!SETSCRAP** and whenever you click it, the system is set up to save SCRAPFILE in the directory where you put **!SETSCRAP**.

Before we get on to the advanced topic of save boxes (little windows which allow you to type in a filename and drag an icon out to a filer) it allows our applications to dump file output as SCRAPFILE without much programming effort. The next program demonstrates.



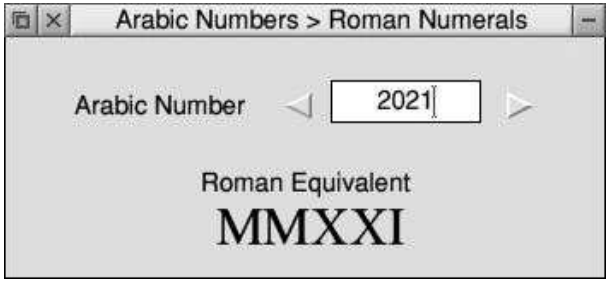

```
ON ERROR PROCERROR
SYS "Wimp_Initialise",200,&4B534154,"Draw Polygons"
DIM B 1000, M 3000, T 1000, V 256, IV(2)
```

8 Demonstration Applications

This chapter puts together all we have learned so far with some fully functioning applications. The standard RISC OS method of quitting an application is via a menu option but since we haven't learned about menus yet, all the applications presented here are quit by pressing the Escape key. (This does have the advantage that it's possible to exit all applications with a single keystroke!)

Roman Numerals

This application lets you convert between Arabic numbers and Roman numerals. It works with numbers 1 to 5000.




```
REM Arabic-Roman
REM The Book of Application Stuff
ON ERROR PROCERROR
DIM B 1000, T 1000, V 50,V(13),R$(12)
RESTORE+1
DATA 1000,M,900,CM,500,D,400,CD,100,C,90,XC,50,L,40,XL,10,X,9,IX,5,V,4,IV,1,I
FOR X=0 TO 12:READ V(X),R$(X):NEXT
Q=FALSE
SYS "Wimp_Initialise",200,&4B534154,"Roman"
PROCWINDOWS :PROCICONS
REPEAT
SYS "Wimp_Poll",,B TO E
CASE E OF
WHEN 2:SYS "Wimp_OpenWindow",,B
WHEN 3:SYS "Wimp_CloseWindow",,B :Q=TRUE
WHEN 6:PROCCLICK
WHEN 8:PROCKEY
WHEN 17,18:IF B!16=0 Q=TRUE
ENDCASE
UNTIL Q OR INKEY-113
SYS "Wimp_CloseDown"
END
:
```

```
SYS□"OS_SpriteOp",&13C,S,,1□:□REM□vdu□o/p□to□screen
ENDPROC
:
```

Insert FNMKWINDOW, PROCMKICON, PROCRDICON, PROCWRICONT, PROCREPORT, PROCERROR

BARCOPY size=5638 bytes, CRC=49879

The structure of Maestro files is detailed in Appendix E (File Formats) of the *Programmers Reference Manual*. Briefly, Maestro files consist of blocks of data, one of these blocks is a table of pointers to tables of events on each channel. Different numbers of channels are assigned to different staves depending how you have set up the Score > Staves in Maestro. An event is either a note or an *attribute*.

A bar line is an attribute (a null followed by &20) and PROCFF (Fast Forward) is used in PROCEXPORT to count bar lines up to the start bar and the channel pointers are noted. A second call to PROCFF 'winds on' to the end bar, again the pointers are noted. The events in between are copied to the end of the relevant event table and the table of pointers is updated to reflect the increased table size.

Draw Utilities

The next few applications manipulate Draw files in some way or another. The structure of Draw files is discussed in *The Book of Draw Stuff*, available from Drag 'N Drop publications, and will not be regurgitated here. Remember that if you're using RISC OS prior to 3.5 you need to set up SCRAPFILE as described in Chapter 7. If mysterious errors appear it proably means the application doesn't have enough memory to hold the Drawfile so increase the value after DIM M, for example DIM M 14000 to DIM M 20000.

Stave Paper

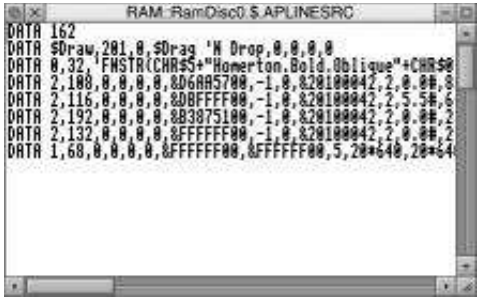
You can print your own musical manuscript paper with this application. Appearance of the staves can be controlled: number of staves, pre-drawn bar lines, width, distance between stave lines and systems and the margins. The icons are pre-populated to generate a standard 12-stave A4 sheet. Click the 'Render Draw' button to export to Draw where you can send to your printer, PDF etc.

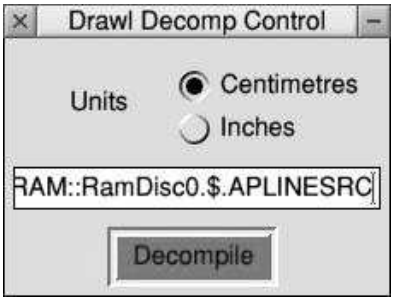
```
REM□Stave□Paper□Version
REM□The□Book□of□Application□Stuff
```

DrawL

DrawL (short for **Draw L**anguage) is a Draw file decompiler. It converts Draw files to a series of DATA statements which can be typed in, designed for magazine listings like *Drag 'N Drop*. DrawL can process font tables, text and path objects. Bitmaps and other types of objects are skipped.

Drag the Drawfile to Drawl's icon bar icon. The filename is appended with SRC (for **S**ou**RC**e) which can be altered if required. Then click the Decompile button to output to Edit. You then need to replace the DATA statements in the second listing (which you can do by drag-saving out of Edit). The example DATA statements produce the mountains picture shown.





```
REM Drawl Decompiler
REM The Application Tutorial And Cookbook
ON ERROR PROCERROR
SYS "Wimp_Initialise",200,&4B534154,"Drawl Decompiler"
DIM B 1000, T 1000, U 1000, V 256, M 20000
B21=1<<21 : Q=FALSE
PROCSPRITES : PROCWINDOWS : PROCICONS
REPEAT
SYS "Wimp_Poll",,B TO E
CASE E OF
```

9 Menus

So far in this book our applications have consisted of icons which the user clicks (or drags files to) to carry out some action. This is an entirely acceptable way for the user to interact with the computer, and some computer platforms rely heavily on this. The downside is the screen can be cluttered with icons.

That's where menus are useful. They're found on other platforms, of course, and on RISC OS they're basically lists of icons laid out vertically. As we know, icons contain text and/or sprites. The user calls up this list of icons (a menu) with the dedicated Menu mouse button, clicks or types some selection, and the menu disappears. There are some RISC OS conventions like clicking select on a sub menu keeps the main menu open but we shall cover these in due course.

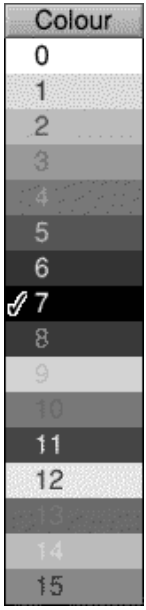
Like windows and icons, menus live in memory blocks and they must be structured in a certain way. Unlike windows and icons, though, there's no help from RISC OS. You don't put the menu definition in the Block and make a SYS call to set it up and get a handle for later use. Instead, we have to know where in memory our menu data is stored and we then pass that to RISC OS to open the menu, on demand.

Because of this bizarre lack of a standard, there are at least many ways of setting up menus as there are RISC OS publications! Essentially menus are in two parts: they have a 28-byte (7 word) Menu Header block and any number of 24-byte (6 word) Menu Entry Blocks. The diagrams on the following pages show what needs to go where. You will see the Menu Entry Block is very similar to the Icon Block and the Menu Entry Icon Flags are identical to those of a normal (window icon) but with some bits ignored (marked X).

Menus and Submenus

The next program demonstrates by allowing you to click the Menu button to bring up a simple menu. Note the application doesn't set up any windows, for demonstration purposes it responds to a menu click anywhere on the desktop and overrides other application's menus (press Escape to quit). Usually you would want to know which window the pointer was on and we will come to that later.

Foreground or > Background. The demonstration also puts a tick against the selected colour CL, which is initially set to 7 (black). PROCTICK(A,X) EORs a one into the menu entry X whose header is at A. PROCMENUSELECT is enhanced with a call to PROCTICK to remove the tick at is old position, set CL to the new value from !B and another call to PROCTICK to mark the tick against the new entry.



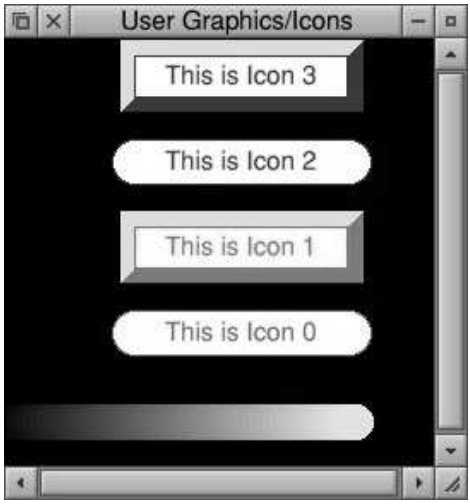

```
ON ERROR PROCERROR
SYS "Wimp_Initialise",200,&4B534154,"Colour Menu"
...

DEF PROCMENUSELECT
I$=STRING$(32," ")
SYS "Wimp_DecodeMenu",,N1,B,I$ TO ,,,I$
PROCREPORT("You chose option "+STR$!B+", '"+I$+"'.")
PROCTICK(N1,CL) : CL=!B : PROCTICK(N1,CL)
ENDPROC
:
DEF PROCMENUS
N1=N
RESTORE +1
DATA "Colour",T,0,0,&70207,160,40,0
PROCMKMENU(N1)
FOR IC=0 TO 15
Y=-(IC=0)*&100-(IC=15)*&80
PROCMKENTRY(N1,28+IC*24, Y,-1, &101+(IC<<28)+((IC EOR 7)<<24), STR$IC, 0)
NEXT
PROCTICK(N1,CL)
ENDPROC
```

isn't like a traditional Basic program where we could draw the circles and leave them there, we have to know the each circle's colour and radius every time we redraw the window, which is why they are set up in the CX, CY and CR arrays at the start and not in PROCPLOT. The icons are set up as before in PROCICONS.

PROCGCOL is used to give a better choice of colours than the 16 desktop ones. It uses SYS "ColourTrans_SetGCOL" which takes as its parameter the 24-bit colour in bits 8-31 of R0 (&BBGGRR00).

Custom Icon Borders



With user graphics we're at liberty to enhance the appearance of icons, as the next program demonstrates. The complete details of the window are obtained with SYS "Wimp_GetWindowInfo". The system returns the number of icons in the window B+88 and the 32-byte (8-word) Icon Blocks are at B+92, B+124, B+156... . PROCBORDER peeks the coordinates of the icon and calls PROCROUND or PROCPLINTH to create icons on plinths or lozenges.

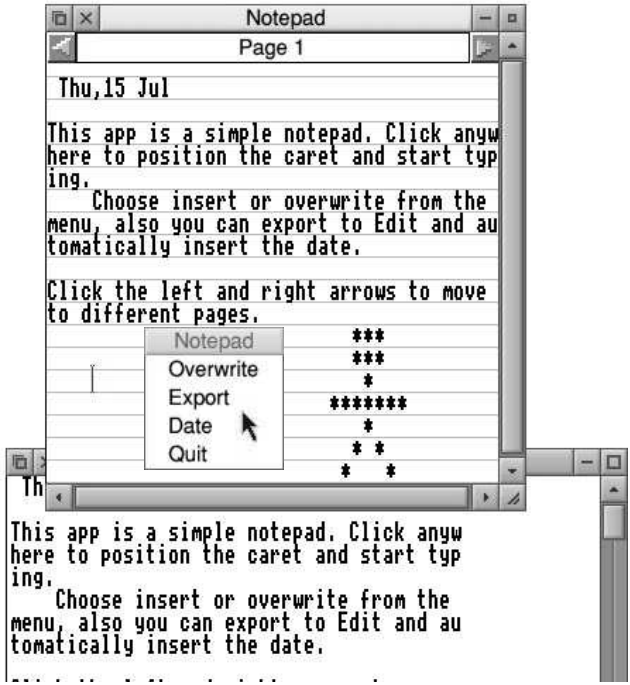
```
SYS"Wimp_Initialise",200,&4B534154,"Windows with user graphics"
DIM B 256, T 256, U 1000, V 100
Q=FALSE
PROCWINDOWS:PROCICONS
!B=W1
SYS"Wimp_GetWindowState",,B
SYS"Wimp_OpenWindow",,B
REPEAT
SYS"Wimp_Poll",,B TO E
CASE E OF
```

The key press is detected with IF E=8 PROCKEY in line 15. PROCKEY retrieves the key code from the Block. These codes aren't INKEY values but (with some exceptions) the Ascii value of the key pressed or special values for control keys. You can use this program to find out the key codes or refer to Appendix 4. If the key in question was 0-9 or A-F PROCCOLOUR deletes the window definition and recreates it with the appropriate background colour.

Any unwanted key presses shoud be passed back to the system with SYS "Wimp_ProcessKey", , K where K is the value originally. retrieved from the Block.

Text and Graphics Editors

The next three listings demonstrate how to implement user graphics for rudimentary text and drawing applications.



Notepad

The next listing is for a simple text editor (notepad). A blank page of feint ruled lines is presented and you click anywhere on the page to position the caret and start typing. You may also use the arrow keys on your keyboard to

```
FOR▫I=0▫TO▫12▫STEP▫4:READ▫I!L:NEXT:▫REM▫scaling▫table
O$="OS_SpriteOp"
SYS▫O$,&109,S▫:▫REM▫init▫sprite▫area
SYS▫O$,&10F,S,"224",,8,8,4▫:▫REM▫create▫sprite▫in▫Mode▫4
SYS▫O$,&13C,S,"224"▫:▫REM▫vdu▫o/p▫to▫sprite
VDU▫23,224,223,223,223,0,251,251,251,0,▫224
SYS▫O$,&13C,S,,1▫:▫REM▫vdu▫o/p▫to▫screen
SYS▫"OS_Byte",20▫:▫REM▫restore▫chars
ENDPROC
:
```

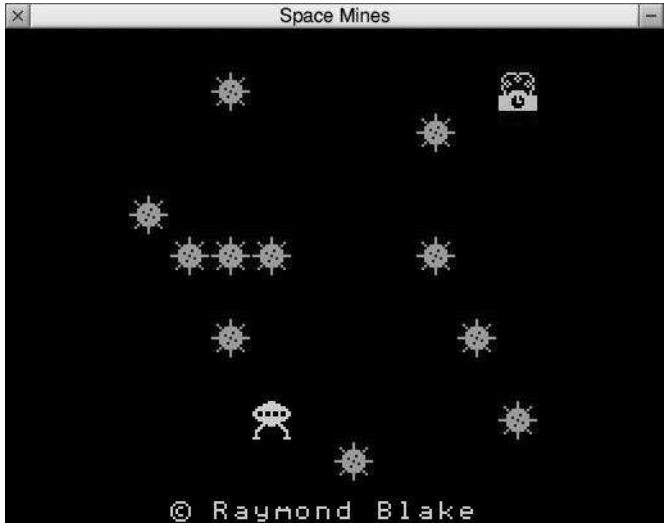
Insert PROCREDRAW, FNMKWINDOW, PROCMKICON, PROCERROR
UG5 size=1869 bytes, CRC=14367

256-colour sprites

If you wish to display a 256-colour sprites then you don't need to manually poke in 256 entries in the 'G' palette table. RISC OS offers a short cut. Use SYS "ColourTrans_GenerateTable", 13, , -1, , G. (13 is Mode 13 and G is the palette table address as before.)

Advanced Topics – 8 Bit Emulation

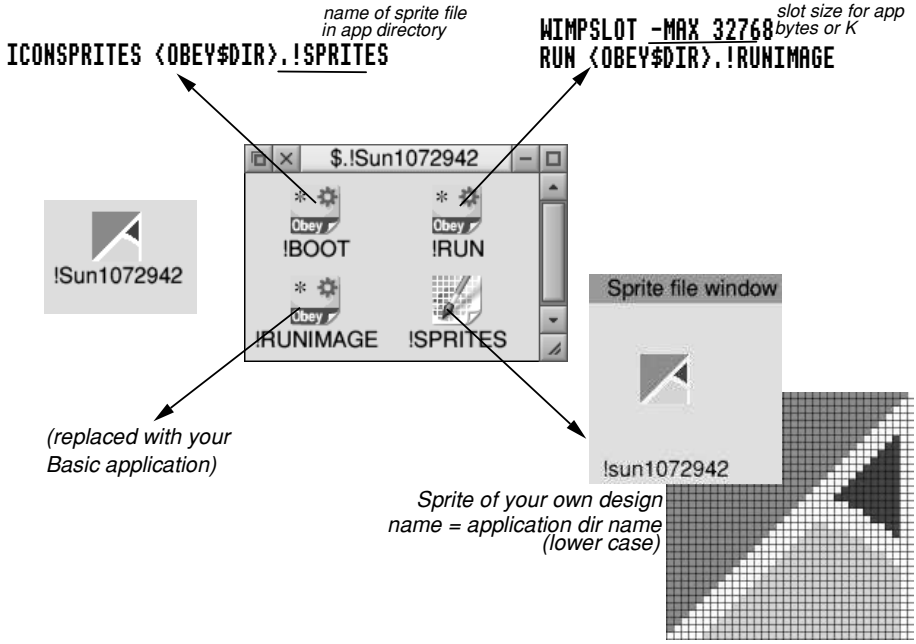
The following listing pulls together topics covered in this chapter with a RISC OS desktop version of a ZX Spectrum game, Space Mines (copyright acknowledged).



11 Application Directories

All of the applications in this book have been self-contained, single listings. This is for convenience and economy: you as the reader only have to worry about one listing at a time and we can maximise the number of demonstrations programs. As you know, RISC OS, applications usually live in special folders, called *application directories*. They're always prefixed with a pling (!) – !Edit, !Paint etc. To open an application directory you hold down the Shift key and double click.

We can get RISC OS to display a different application icon, in the filer, to reflect what each application does. In Chapter 6 we used a small obey file to set a more appropriate slot size and in Chapter 8 the label designer needed a directory to store label definitions. In both cases the extra data had to be stored, rather awkwardly, in the CSD. If they are kept privately, out of sight in the application directory there's no need to worry about manually setting the CSD for each application.



12 A General-Purpose Save-As Dialogue Box

A situation frequently arises on RISC OS where we need an app to respond to a file dragged to it, perform some operation on it, and output a result such as a converted file.

A style has evolved in RISC OS where saving a file from an application is done through a *dialogue box*. This is a small window whose title is usually "Save as" and the window has a minimum of three icons – a file icon to illustrate the file type of the file being saved, a writeable icon which starts out as a suggested filename (Document1, File1 etc.), or contains the full pathname of a previously-saved file, and finally an OK button.

The user can do one of three things to save the file. He or she can drag the file icon to a directory viewer, press Return or click the OK button. The last two actions are analogous to the "Save" option on other computing platforms. However the filename displayed in the writeable icon must be a full pathname otherwise an error message is usually shown "Please drag the icon to a directory viewer" or suchlike.

The "Save As" appears either as a sub menu (sliding right off a "Save" menu option) or after dragging a file to the application's window or icon bar.

Unfortunately, the operating system provides no help in setting up this window, despite its uniquity on RISC OS. (The Toolbox modules provide some assistance but that is beyond the scope of this book.) So we have to pull together our knowledge from Chapters 4, 5, 7 and 9 and be warned – it's rather tedious! The listing below demonstrates and you can use it as a template for your own routines.

Appendix 2
System Calls

This appendix lists the SYStem calls used in the programs in this book, with a brief summary only. Advanced information is available in the *Programmer's Reference Manual*.

SYS "DragASprite_Start",X,Y,"NAME",B

Initiate dragging sequence. X options include 0 for standard dragging sprite **"NAME"** as a greyed-out (dithered) sprite anywhere in desktop (save operations) or &10 to limit to within window. Other parameters in Block at B.

SYS "Font_FindFont", , "Name", X, Y TO F1

Open a font for use, returning handle in F1. "Name" is font already in system eg Trinity.Bold and X and Y are horizontal and vertical point sizes × 16 (192 is 12 pt).

SYS "Font_ListFonts", , B, C, 256 TO , F$, C

Returns a font name installed in the system, B is the workspace, C is TRUE if the end of the font list has been reached, FALSE otherwise, F$ is the font name.

SYS "Font_LoseFont", F1

Close font (previously opened with Font_FindFont) whose handle is F1.

SYS "OS_Byte", A, X, Y

Osbyte call (a hangover from BBC Micro) equivalent to *FX A,X,Y where A=Osbyte number and X, Y are the registers.

SYS "OS_File", 5, F$ TO , , , , Z

Gets file size (bytes) of filename whose path is in F$ and stores in Z (note quadruple comma before Z)

SYS "OS_File", 8, F$

Creates a directory (in the CSD) with filename **F$**.

SYS "OS_File", 10, F$, T, , M, N

Saves block of memory to file **F$** of type **T** starting at **M** and ending at **N** Note double comma after T.

Appendix 3

Hex Binary

Hex	Binary	Hex	Binary
0	0000	8	0000
1	0001	9	0001
2	0000	A	0000
3	0011	B	0011
4	0000	C	0000
5	0001	D	0001
6	0000	E	0000
7	0111	F	1111

Logic Tables

The old value is the (hex) number in the leftmost column. The EOR, OR or AND value is at the top of the next columns and the intersection is the new value. For example, &8 EOR &7 = &F.

EOR

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	0	3	2	5	4	7	6	9	8	B	A	D	C	F	E
2	2	3	0	1	6	7	4	5	A	B	8	9	E	F	C	D
3	3	2	1	0	7	6	5	4	B	A	9	8	F	E	D	C
4	4	5	6	7	0	1	2	3	C	D	E	F	8	9	A	B
5	5	4	7	6	1	0	3	2	D	C	F	E	9	8	B	A
6	6	7	4	5	2	3	0	1	E	F	C	D	A	B	8	9
7	7	6	5	4	3	2	1	0	F	E	D	C	B	A	9	8
8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
9	9	8	B	A	D	C	F	E	1	0	3	2	5	4	7	6
A	A	B	8	9	E	F	C	D	2	3	0	1	6	7	4	5
B	B	A	9	8	F	E	D	C	3	2	1	0	7	6	5	4
C	C	D	E	F	8	9	A	B	4	5	6	7	0	1	2	3
D	D	C	F	E	9	8	B	A	5	4	7	6	1	0	3	2
E	E	F	C	D	A	B	8	9	6	7	4	5	2	3	0	1
F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

Glossary

Acorn Small computer company based in Cambridge, UK, from the 1970s to the 1990s. Produced the world's first 32-bit, Arm-based desktop computer (the Archimedes) after success with the BBC Micro.

Ampersand &. Used to indicate hexadecimal on RISC OS, eg &12349ABC. Prefix. Other systems use different nomenclature for example 0x as in 0x12349ABC.

Archimedes World's first desktop computer released in 1987 using an Arm microprocessor. Famed for its speed and greenness owing to low power consumption.

Arthur Operating system found on Archimedes and early 'A' series machines of the 1980s. Sometimes called RISC OS 1 as it was the first version of the OS before the term "RISC OS" had been invented (the first 'official' RISC OS release was RISC OS 2).

Arm Microprocessor chip built into the Archimedes desktop computer, subsequently powering RISC OS and millions of devices worldwide including mobile phones Raspberry Pi's.

BBC Micro 8-bit computer with a 6502 microprocessor produced by Acorn starting in 1981. Given the BBC branding when it was chosen to be the machine for an early UK Government drive towards computer literacy. Spawned several models including the BBC B, Master and was the forerunner to the Archimedes.

Bit Smallest unit of computer memory, either 0 or 1 (off or on, unset or set).

Block Area of memory or parameter block requiring data to be present in a certain order before making a SYS call to get the system set up a window, icon etc.

Byte Unit of computer memory equal to eight bits.

Clear Word Value ANDed (usually after being inversed with NOT first) with a word to reset bits in preparation for setting bits with an **EOR Word**.