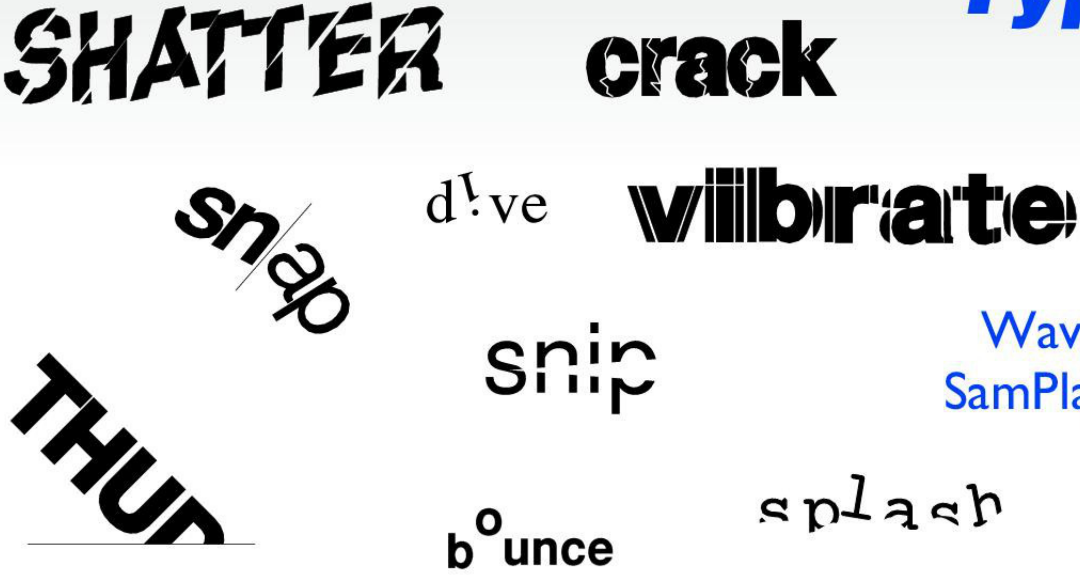
DRAG'NDROP

for RISC OS computers

Volume 13 Issue 2
Winter 2024
£4.25

Type Play



WaveSynth samples

SamPlay – type in app

RockPro 64

Noah's Arc

Wakefield 2024

Acorn & RISC OS Computer Show

Saturday 27th April

RISC OS AWARDS

Best show or event 2017 and 2018

The North's premier RISC OS show since 1996

RISC OS Awards' best show or event in 2017 & 2018

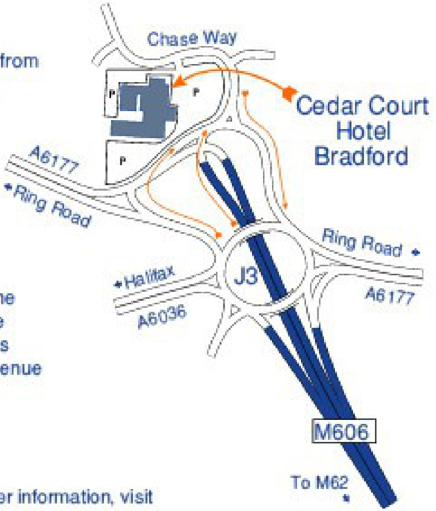
- Big Names and Small Developers
- Raspberry Pi
- Show Theatre
- On-site Catering
- Prize Draw
- Charity Stall raising funds for Wakefield Hospice

For 2024, we're remaining at our new venue

The Cedar Court Hotel, Bradford
Mayo Avenue, Rooley Lane, Bradford
West Yorkshire BD5 8HW

(Adjacent to the terminus of the M606, just off the Bradford Ring Road)

Getting there by car
- Easy access via the M606 from M1, M62 and major routes
- Free on-site parking



Getting there by train
- Connections from across the UK to Bradford Interchange and Forster Square stations
- Local buses hourly to the venue

• Date	Saturday, 27th April 2024
• Opening time	10.30am to 4.30pm
• Ticket price	£5 on the door Entry for children aged 12 or under (accompanied by an adult) is FREE
• Access	The show venue is on the ground floor of the hotel
• Enquiries	Wakefield Show 2024, c/o 3 Riverdale Avenue, Stanley, Wakefield, WF3 4LF
• Email	show2024@wakefieldshow.org.uk

For further information, visit

www.wakefieldshow.org.uk

Show organised by

Wakefield RISC OS Computer Club

DRAG 'N DROP

Copyright ©2024 Drag 'N Drop. All rights reserved.
Produced on RISC OS computers

This issue has been blessed with contributions from the following people:

Norman Lawrence (RockPro 64)
Richard Baraniak (RISC OZ)
Christopher Dewhurst (everything else)

Programs have been tested on RISC OS 5 running on the Raspberry Pi. Compatibility with other versions of RISC OS is not guaranteed.

The views expressed in this magazine are not necessarily those of the editor. Alternative views are always welcome and can be expressed by either writing an article or a short editorial.
All articles and advertisements are published in good faith. No materials in this publication are meant to be offensive or misleading. If you come across something you believe is either of the above please contact the editor using the details below.

Contact Information
Editor: Christopher Dewhurst
Email: editor@dragdrop.co.uk
www.dragdrop.co.uk

Contents

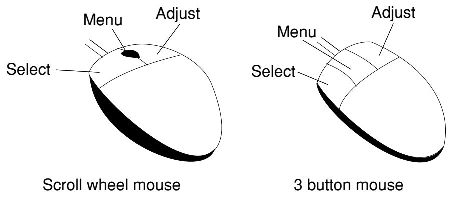
Editorial

Happy new year and welcome to what is a bumper edition of *Drag 'N Drop*, in no small part due to our contributors – special thanks to them. I don't know about you but I'm noticing more developers returning to the RISC OS scene as they realise it is alive and not just a hobbyist platform. Please give them your support. The ever green (in more ways than one) RISC OS is attracting new people too. Three RISC OS shows have been announced so far this year, get along to one if you can!

Done with my new tablet

Chris

RISC OS mouse

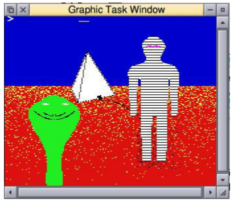


Selectc click (or just click) means click the left button, 'menu over' means click the middle button.

Entering listings

You can type in programs in a number of ways:

1. Very simple prigrams can be entered in a task window, menu over Raspberry icon Task Window ^F12. Cursor keys and graphics will not work.
2. With a text editor like Edit, in Apps folder on the iconbar. Select double click, menu over Edit's iconbar icon, Create>BASIC. For other languages type in at the bottom. Save with F3 and double click to run.
3. Use the built-in editor (single tasking). Press F12, type Basic <return> then EDIT <return>. Refer to Chapter 26 of the BBC Basic Reference Manual free at riscosopen.org.



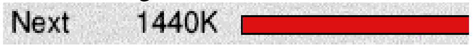
Program lines are numbered when using. EDIT but it can be used in GraphTask armclub.org.uk/free/. You can enter programs withoiut EDIT and run Basic programs that use simple graphics (not sprites) in a window on the desktop.

Program lines aren't numbered in *Drag 'N Drop* listings, it is assumed you use Edit. Each line starts with space so you know when to press Return. To find out the line number you are on press f5 in Edit.

```
 PROCWRICONF(W1,6,10<<28,15<<28): PROCWR
ICONT(W1,6,"Process")
```
Start of line has a space
Continuation of line (no space)

More memory for applications

You may need to reserve more memory for a program. Adjust-click on the Raspberry icon and under Application tasks click and drag the Next slide bar out to the right.



What does 'currently selected directory' mean?

Articles may tell you to set the CSD (currently selected directory). Click menu over filer window and choose Set directory ^W. It's where the computer stores the file when you type SAVE "myprog".

What's an Applcation Directory?

Application directories begin with a ! called 'pling', for example 'pling boot'. To open (without running) it, hold down the shift key and double click select to open the directory.

MODE problems

If you get a ’Screen mode not available’ or blank screens try one or more of the following:

1. Install the Anymode module from pi-star.co.uk/anymode, place it in !Boot.Choices.Boot. Predesk.
2. Hold down Shift and double click !Boot, in the root directory of the SD Card, that is SDFS::RISCOSPi.$.!Boot. Locate the Loader file and with Shift held down double click it to open it. Create a text file in Edit with the following line (press Return at the end):

```
disable_mode_changes
```

Save it inside Loader as CMDLINE/TXT and restart your machine.

3. Install ADFFS from forums.jaspp.org.uk/forum/viewtopic.php?t=4831.
4. Aemulor can interfere with screen modes. Menu over iconbar > Quit > Emulator too.

Sounds are strange

Some listings need the free RDSP module installed. Download it from www.amcog-games.co.uk/rdsp.htm where you’ll find instructions on how to install it.

WIMP library

Many programs in *Drag ’N Drop* run in a window on the desktop). They use a set of standard procedures to create and deal with windows, icons and menus.

Rather than publish them with every listing they are collected here. Most of them are taken from The Application Tutorial and Listings Book available from Drag ’N Drop Publications. If you’re interested in writing desktop applications then you should consider buying this book.

```
DEF FNMKWINDOW
READ $T,X,Y,W,H
FOR I=0 TO 84 STEP 4
READ A$
I!B=EVALA$
NEXT
T+= LEN $T+1
SYS "Wimp_CreateWindow",,B TO X
=X
```

Basic’s DATA pointer is assumed to be at a line giving window title, position, size, colours, flags comes before this is called. Memory blocks T and B must have been set up. Makes a window returning handle in 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 I
ENDPROC
```

Make an icon, handle is returned in I. H=window handle, X,Y=bottom left, W,D=dimensions, F=flags, A$=text and V validation string (1 if none)

```
DEF PROCRDICON(W,H)
!B=W : B!4=H
SYS "Wimp_GetIconState",,B
A$=$(B!28) : X = B!24
ENDPROC
```

Read icon W in window H text in A$ and flags in X.

```
DEF PROCWRICONT(W,H,B$)
PROCRDICON(W,H)
B!8=0 : B!12=0
$(B!28)=B$
SYS "Wimp_SetIconState",,B
```

```
ENDPROC
```

Updates text B$ in icon H in window W.

```
DEF PROCWRICONF(W,H,X,Y)
!B=W : B!4=H : B!8=X : B!12=Y
SYS "Wimp_SetIconState",,B
ENDPROC
```

Updates icon W's flag in Window H, X is the EOR word and Y the clear word, ie flag = (flag AND NOT X) EOR Y.

```
DEF PROCMKMENU(A)
READ $T
FOR I=0 TO 24 STEP 4
READ A$ : I!A=EVAL A$
NEXT : T+= LEN $T+1
ENDPROC
```

DATA line before this is called with menu header details. Sets up menu header at memory address A.

```
DEF PROCMKENTRY(W,X,Y,H,F,A$,V)
$U=A$
RESTORE+1
DATA Y,H,F,U,V,LEN $U
FOR I=0 TO 20 STEP 4
READ B$
I!(W+X) = EVAL B$
NEXT : U+= LEN $U + 1
ENDPROC
```

Basic DATA line before this is called giving menu item details. W=header address, X=offset (multiple of 24), Y=work flags, H=submenu pointer (–1 if none). F, A$ and V as for PROCMKICON. Add entry for menu already set up.

```
 DEF PROCMKSPRITE
 READ A$,W,H,MD,PW,NC
 SYS "OS_SpriteOp",&10F,S,A$,-(NC<>0),W,
H,MD
 SYS "OS_SpriteOp",&125,S,A$,-1 TO ,,,,C
 IF NC FOR X=0 TO NC*8 STEP 8:READ C!X:
NEXT
 READ J$: J=EVAL("&"+J$): K=PW
 FOR Y=H-1 TO 0 STEP -1: FOR X=0 TO W-1
 SYS "OS_SpriteOp",&12A,S,A$,X,Y,J
 J=J>>(32 DIV PW): K-=1 : IF K=0 K=PW: R
EAD J$: J=EVAL("&"+J$)
 NEXT: NEXT
 ENDPROC
```

Create a sprite. DATA pointer must be be at a line giving sprite's name, width, height, mode, pixels per word (eg 32 for 2-colour) and number of colours before PROCMKSPRITE called. If number of colours >0 the following words give RGB palette entries. Shape DATA then given as words (without &) in compacted format. Intended for use with 2-, 4- and 16-colour sprites.

```
 DEF PROCPROGINFO
 RESTORE +1
 DATA About this program,0,0,500,200,X,Y
,X+W,Y+H,0,0,-1,&84001012,&1000207,&C010
3,0,-H,W,0,&13D,0,1,0,T,0,0,0
 W0=FNMKWINDOW: $V="R2"
 DATA Name,Purpose,Author,Version
 DATA Application Demo,Drag N Drop,n.m (
dd-mm-yyyy)
 FOR IC=0 TO 7
 READ A$:LOCAL DATA
 IF IC<4 PROCMKICON(X,0,-50-IC*50,130,50
,&97000301,A$,0)
 IF IC>3 PROCMKICON(X,140,-50-(IC MOD4)*
50,350,50,&9700010D,A$,V)
 RESTORE DATA
 NEXT
 ENDPROC
```

Create 'About this program' window giving it handle W0. Adust 3rd line of data as needed. Needs FNMKWINDOW and FNMKICON.

Application Directories

Type-in applications for RISC OS in the magazine are usually presented as single listings in BBC Basic for convenience and space reasons. The standard 640k of memory allocated under Next in the Tasks window is more than enough for most Drag 'N Drop applications.

The programs can be put into proper RISC OS application directories as follows. Create a new directory whose name begins with a pling (!) followed by the application name eg !Txt2Draw. Hold down shift and select double click to open it.

Copy the application typed in from Drag 'N Drop into the application directory. It is conventional to rename this main program !RunImage but it isn't compulsory.

Either run the program below or create your own application sprite, it should be no more than 68×68 pixels big. Ensure the sprite name is identical to the application directory name (but in lower case) and save it inside the application directory.

```
 DIM S 210
 !S=210:S!8=&10:SYS "OS_SpriteOp",&109,S
 DATA !myapp,32,32,25,32,1, &BFE3F200,&A
8540000
 DATA 0,0,BC383FBF,FE3C7F7F,EF7C7776,C76
E77E6,76E3FE6,F7FF3FE6,EECF7776,FFCFF77F
,7DE7EFBF,0,0,F7C00,66C00,6E800,7E400,76
000,66000,6F000,0,7F493FBF,FE3E7F7F,EE63
7776,EE4177E6,FEC1FFE6,7E413FE6,1E637776
,1E3EF77F
 DATA 3F49EFBF,0,0,0
 PROCMKSPRITE
 SYS "OS_SpriteOp",&10C,S,"!Sprites"
 END
```

Next create two Obey files in the application directory, one called !Boot with the following line

```
iconsprites <obey$dir>.!sprites
```

and a second Obey file called !Run with the following lines:

```
wimpslot -min 32K -max 32K
run <obey$dir>.!RunImage
```

the K (kilobyte) numbers after wimpslot command may need to be adjusted depending on how much memory the application needs.

Now simply double click the application to run it.

!MyApp

Have you a story to tell about your RISC OS computer or even written a program for it?
Send it to dragdrop@dragdrop.co.uk. Payment available for quality RISC OS journalism.

News and Apps

RISC OS Shows Galore

2024 looks to be a bumper year for RISC OS exhibitions. The South West show takes place in Bristol on Saturday 24th Feburary, doors open 10.30 and admission is only £6. More details at riscos-swshow.co.uk.

In the spring there's the Wakefield RISC OS show on Saturday 27th April. The venue is the Cedar Court hotel in Bradford owing to its sister hotel in Calder Grove's continued use for housing asylum seekers. See wakefield-show.org.uk for further details.

And after a two year hiatus the RISC OS London Show returns on 26th October in Harrow. Click along to riscoslondonshow.co.uk to find out more.

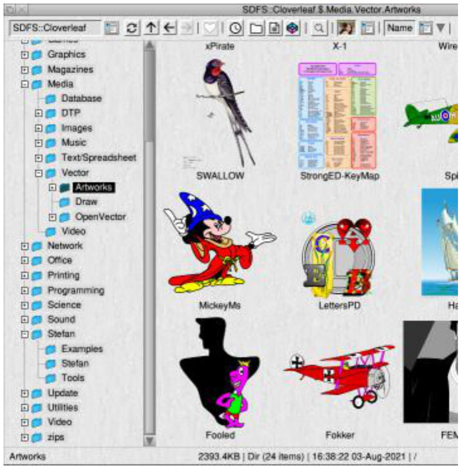
New venue for RISC OS London show (image: ROUGOL)

Netsurf 3.11

Netsurf is a free, open-source web browser for several OS's and whilst it lacks the heavyweight code allowing you to do your online banking (use Iris for that) it's still the preferred choice for basic browsing because it is compact and fast loading. The huge number of updates for the package include improved SVG handling, bitmaps, text areas, text selection and the ability to switch CSS off. Just visit netsurf-browser.org to update your copy.

TCP/IP Stack 7.04

Not a solution for sore throats, rather in the computing world a 'stack' of TCPs (Transmision Control Protocol) and IPs (Internet Protocol) is a standard for computers sending messages, the basis of our internet browsing and most importantly bringing true wi-fi to RISC OS. Read more at riscdev.com.



CLFiler – modernises RISC OS and costs less than £15

CLFiler 1.55

Stefan Froeling continues to develop the feature-rich RISC OS filer, an state of the art alternative to the elderly one built into RISC OS which, let's face it, looks very retro nowadays. CLFiler costs 15,97 euros (£13.60) from riscoscloverleaf.com with regular updates. CLFiler runs alongside the usual RISC OS filer so you don't even have to give up habits to try it out.

BeebIt 0.76

Those who take their diet of retro 8-bit software on RISC OS should download an update to the BBC Micro emulator, BeebIt. The new version fixes display issues such as a smoother Mode 7 font and ensures Beebit runs on the latest RISC OS hardware like the PineBook. Get along to mjfoot.netlify.app/bbc.htm.

● 8-Bits will be back next time.

Pluto 3.20

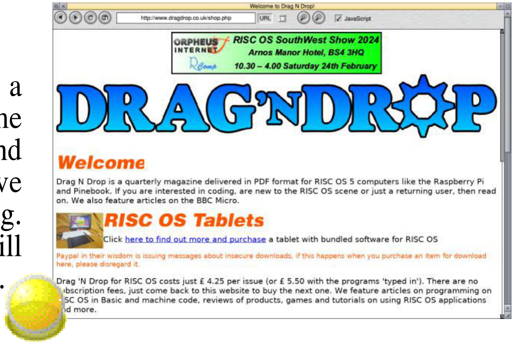
Pluto is a feature-rich freeware news and email client (named after the dog who fetched post) and is up to version 3.20, the update mostly streamlines the app by cleaning out redundant code. Download your copy from www.avisoft.f9.co.uk.

Timerun

The latest offering from Amcog is a vertical-scrolling shoot-em-up game featuring high standard graphics and musical accompaniment which we've come to expect from the Amcog. Timerun costs £9.99 via !Store. It will be reviewed in the next *Drag 'N Drop*.

Origyn

The noughties and '10s spawned open source, cross-platform, web browsers – Otter, Qupzilla and Origyn to name a few – based on the Webkit engine used by Macs. Many retired into oblivion but Origyn is still popular on AmigaOS and Michael Grunditz has been porting it to RISC OS. It's in very early stages but you can show your support by emailing Michael at micken@dfupdate.se and help with testing. A sunshine yellow could be a regular feature of your iconbar!


Welcome to Drag N Drop!

http://www.dragdrop.co.uk/shop.php

RISC OS SouthWest Show 2024
Arnos Manor Hotel, BS4 3HQ
10.30 – 4.00 Saturday 24th February

DRAG'N DROP

Welcome

Drag N Drop is a quarterly magazine delivered in PDF format for RISC OS 5 computers like the Raspberry Pi and Pinebook. If you are interested in coding, are new to the RISC OS scene or just a returning user, then read on. We also feature articles on the BBC Micro.

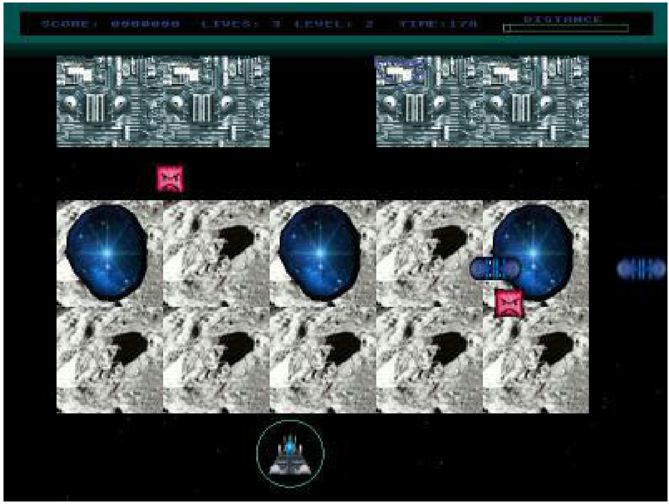
RISC OS Tablets

Click here to find out more and purchase a tablet with bundled software for RISC OS

Paypal in their wisdom is issuing messages about insecure downloads, if this happens when you purchase an item for download here, please disregard it.

Drag 'N Drop for RISC OS costs just £ 4.25 per issue (or £ 5.50 with the programs 'typed in'). There are no ubscription fees, just come back to this website to buy the next one. We feature articles on programming on SC OS in Basic and machine code, reviews of products, games and tutorials on using RISC OS applications d more.

New from Amcog – Timerun (Image: Amcog)



Type Play

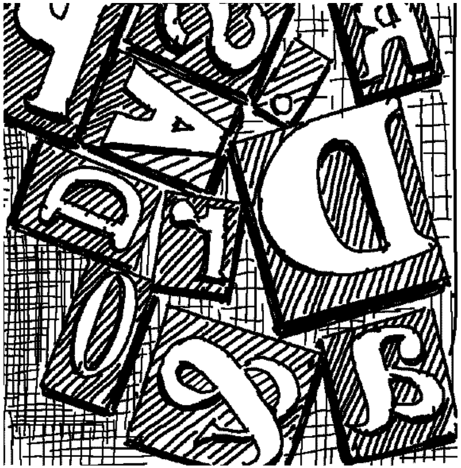
Ever since the advent of the printed word and moveable type, people having been playing with fonts and arranging letters from typefaces in an amusing way.

Invention of rub-down transfers like Letraset took this to a new level and with the advent of computer fonts people went to a messy extreme with complete trashing of fonts. The original art form, however, was like a game with a few simple rules:

- ● Letters must be from the original typeface, variants (heavy, italic etc.) are allowed.
- ● Letters may not be distorted other than by translation, rotation and slicing.
- ● No other graphics (other than empty space) can be used.

This is very easy to do in Draw on RISC OS, the vector drawing package which has come free with every version of the operating system.

We're going to do some 'onomatopoeic' type play, where the result is a display looking like it sounds, to reflect the articles on sound sampling elsewhere in this issue of *Drag 'N Drop*.

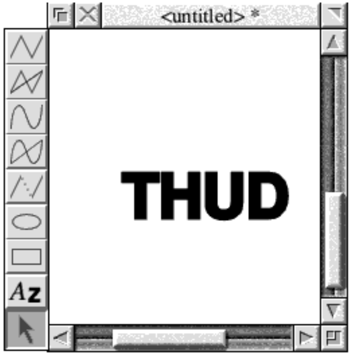


Simple Type Play

Open a new Draw document. Select the text (AZ) icon from the toolbox on the left. Click Menu > Style > Font Name > Homerton > Bold. Use the Adjust button so that the menu remains on the screen. Move the pointer back to the Style menu choose Font size. Replace the 6.40 at the bottom with 72 (72 points or one-inch).

The RISC OS Select button is better than other platforms, you would either have to navigate through the whole menu structure again or select the type size and style from a dialogue window cluttering your work area.

Click anywhere on the Draw document and type THUD. Click on the pointer icon in the toolbox and select the THUD word. Menu > Select > Convert to Path, move back to the Draw menu and Style > Line width > 4 amd Style > Line colour > and choose black from the top right of the palette.



Easier Menus

When it comes to programming menus in Wimp applications there is an inconsistency with the RISC OS window manager. SYS "Wimp_Create Window",,B creates a window for later use, with parameters given in the block (B). Since there is the word 'create' in the call you might expect SYS "Wimp_CreateMenu",,B to do the same for menus – initialise a block, return a handle for later use with SYS "Wimp_ OpenMenu",,B.

Except it doesn't. SYS "Wimp_ CreateMenu",,B in fact tries to display a menu from a block which the user is expected to have already set up, by poking data into memory – precisely and correctly otherwise the sytem seizes up.

Apparently, in the original Acorn Archimedes operating system menus were put in to the window manager at the last minute. This would explain why things work as they do – why there's no SYS "Wimp_Open Menu" ,,B.

The result of all this is that the programmer has been left 'in the wild' to devise ttheir own way of building the menu block.

In BBC Basic this would typically involve passing long strings to a massive procedure which parses it using MID$, LEFT$, RIGHT$, special characters like apostrophes, asterisks etc. representing ticks, submenus and other flags of menu entries.

There's certainly no standard way of going about this and Chris Dewhurst says in *The Application Tutorials and Listings Book* there are about as many ways of encoding menu data as there are listings for RISC OS! He opts to set menus up in a simple manner with just a couple of standard procedures. Menu text, colours and flags are spelt out in full for each item.

Regardless of the method you use, menus are a pain and whilst I found the *TATALB* approach refreshing, I still thought there was something I could do make the process less cumbersome.

At the same time I didn't want to constrain the programmer, by insisting all entries are a uniform colour, for example, or every entry has to be text.

Firstly I decided that all menu details are stored as DATA statements, not just for PROCMKMENU and use a new procedure for setting up the menu headers (title bar) which I will list shortly.

I standardized the data into seven items, for both menu headers and menu entries. I'm assuming you have a copy of *TATALB* to hand so you can follow my working in conjunction with Chapter 9.

Param	Headers	Entries
0 X	Instructions	Instructions
1 H	Colours	Workflags
2 W	Width of entries	Submenu/ window addr
3 F	Height of entries	Iconflags
4 A$	Title text	Entry text
5 V	Validation (0)	Validation
6 P	Pointer add	Pointer add

The Instructions parameter tells the program what to do with the following

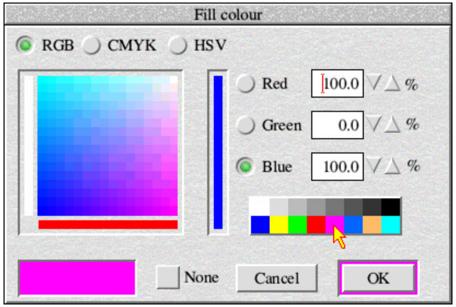
The majority of windows and icons on the RISC OS desktop use a standard palette of 16 Wimp colours (eight greys and eight colours) which have remained the same throughout every release of the operating system.

They were carefully considered as being suitable for applications when RISC OS was introduced without being too intrusive for desktop work. In practice about 75% of the shades appear and for some applications the default palette may not be suitable for people with visual disabilities or just too limiting.

It's possible to customise the Wimp palette using SYS "Wimp_SetPalette",, B where B is a 20 word block giving the 16 colours. 3 mouse colours and border in &B0G0R000 format (eg &00F0F000 is red). Individual colours cannot be set, the whole palette must be read using "Wimp_ReadPalette",,B, colour(s) amended and the block written back. The border colour appears to be redundant in RISC OS 5.

The demonstration program in Listing 1 changes the mouse pointer to yellow with a red border and makes colours 8-15 brighter shades, which can be useful for Draw.

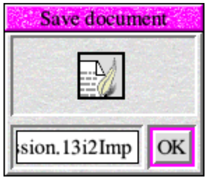
Incidentally, to get the default Wimp palette simply issue the call SYS "Wimp_SetPalette" with no parameters.


Fill colour

RGB [✓] CMYK HSV

Red 100.0 %
Green 0.0 %
Blue [✓] 100.0 %

Customising the Wimp palette opens up a wider colours in Draw...



...but if not chosen carefully can result in garish colours in other apps

Listing 1

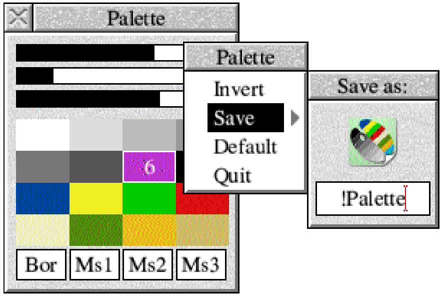
```
REM adjusting the Wimp palette
REM (c) Drag N Drop Winter 2024
DIM block 20*4
SYS "Wimp_ReadPalette",,block
block!(8*4)=&F0000000:REM blue
block!(9*4) =&00F0F000:REM yellow
block!(10*4)=&00F00000:REM green
block!(11*4)=&0000F000:REM red
block!(12*4)=&F000F000:REM magenta
block!(13*4)=&F0660000:REM dk orange
block!(14*4)=&60B0F000:REM salmon
block!(15*4)=&F0F00000:REM cyan
block!(17*4)=&0000F000:REM mouse 2
block!(18*4)=&00F0F000:REM mouse 3
block!(19*4)=&0000F000: REM border
SYS "Wimp_SetPalette",,block
```

The RISC OS Style guide seems to have caused programmers to develop an allergy to the palette, few colours and only four or five of the greys are used, The remainder can be freely changed without resulting in a garish desktop.

● Next time in *Drag 'N Drop* we'll present a desktop palette editor.

Next time in Drag 'N Drop...

Applications

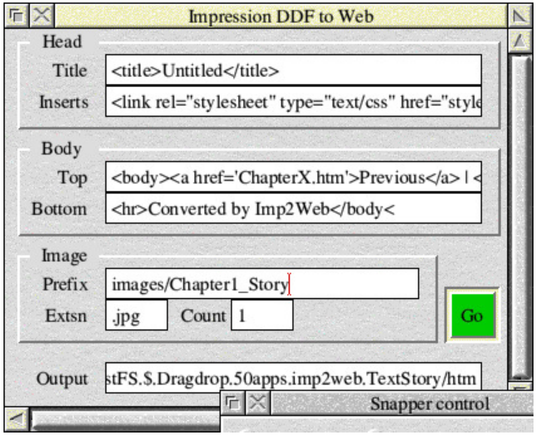


Palette

Remember that desktop palette editor in previous versions of RISC OS? We don't know why it was taken out either but you can have it back with our type-in app.

Impression to HTML

Convert your Impression documents for web browsing with ease. Full listing.


Impression DDF to Web

Head

Title <title>Untitled</title>

Inserts <link rel="stylesheet" type="text/css" href="style

Body

Top <body><a href='ChapterX.htm'>Previous</a> | <

Bottom <hr>Converted by Imp2Web</body<

Image

Prefix images/Chapter1_Story

Extsn .jpg Count 1

Go

Output stFS.$.Dragdrop.50apps.imp2web.TextStory/htm

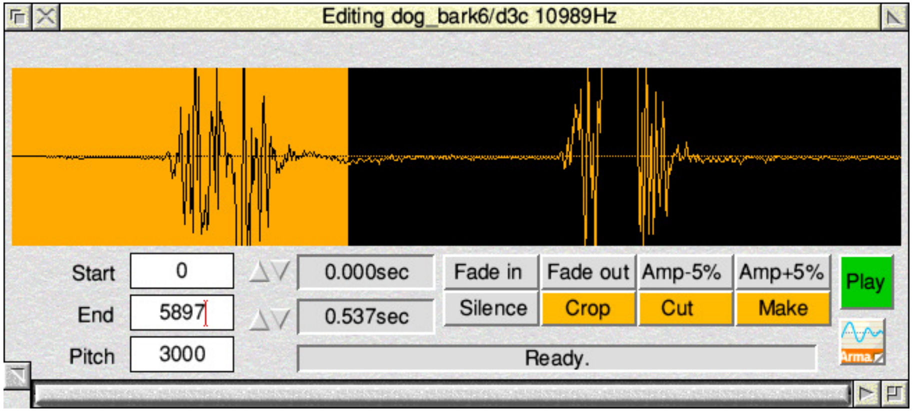
Snapper control

Noah's Arc

Continuing our delve into the past while learning all about programming your computer.

. . . Plus all your faves!

SamPlay



SamPlay is an application which can play sound samples of type &b3c (Armadeus) format, an audio standard on RISC OS. It's written in BBC Basic and provides editing functions like fading, cropping and cutting normally only found in commercial software.

Positions within the sample can be selected with precision, down to sample number.

Edited samples may be saved for processing – see the article on playing samples with WaveSynth elsewhere in this issue of *Drag 'N Drop*.

Once you have typed in the listing and fully debugged it, double click it and SamPlay will park its orange icon on the iconbar. A window will open and depending on how much memory was in the Next slot, the maximum sample size will be displayed at the bottom.

Drag an Armadeus file onto the SamPlay window and its waveform will be shown in orange on black. The window tiutle bar will show the filename plus sample rate in Hz.

Clickung and dragging on the waveform will select (invert) the selected part of the waveform. Exact points in the sample may be selected by bumping the arrows next to Start and End, or typing in the writeable icons. The equivalent time in seconds to 3 d.p. will update in the boxes to the right.

The group of eight icons on the near right apply effects to the selection. For example, fade out will gradually reduce the amplitude to zero over the specified period.

Three of the icons are coloured orange indicating caution. Crop takes out the selection and discards the rest, displaying the cropped as a new waveform. Cut has the opposite effect, the selection is discarded and parts either side spliced to make a new waveform.

If the sample has just been loaded or edited, click Make to make up the sample in the internal format required by RISC OS then click Play. Just click Play after that. Alter the pitch by adjusting the number next to Pitch, which the 16-bit pitch in Basic's SOUND statement in hex - &3000 is the C below middle C.

Clicking the Armadeus icon bottom right will save the sample using an incremental serial number, for example if the file is BOOM, clicking once will save it as BOOM1, then BOOM2 etc. If

WaveSynth Samples

The WaveSynth module built into every version of RISC OS is the one which plays the 'beep' sound when your computer starts up and usually accompanies errors displayed on the desktop.

It's also the default sound when you type SOUND 1,-15,53,20 or similar in BBC Basic. You can see this when you type *voices. At the top you see channel 1 has been assigned to voice 1, WaveSynth-Beep.

WaveSynth is capable of much more than providing this rather pathetic beep, however, and one of those capabilities is to play sound samples. By the end of the article you can play all those dog barks and laser gun effects you downloaded from the internet, or even sampled yourself, to accompany your games and presentations.

You don't need to go to the complexity of writing machine code voice generators, WaveSynth can do it for you. All you need to know is how to make new 'wave tables'.

A wave table is a file telling RISC OS not only what sample to play (the wave data) but also how to play it. To do this we get WaveSynth to create a sort of sibling of itself (called an 'instantiation') and you can then 'play' your samples with the regular BBC Basic SOUND statement.

The *Programmer's Reference Manual* devotes just four pages to the workings of WaveSynth, most of it taken up by an abstruse listing. The source code for WaveSynth at www.github.com isn't much clearer.

To begin to understand it we need to cover some digital sound theory, it's not that scary and I won't go into too much detail.

When the computer plays a sound, it sends a byte of sample data to the speaker every 48 millionths of a second, which is to say the sample period is 48 microseconds.

Looked at another way, 1,000,000 divided by 48 = 28,033 bytes are send to the speaker every second.

This *sample rate* is what you read about in digital sound literature quoted in hertz (Hz) or kiloHertz (kHz). You divide into a million to get back to the sample period, eg 1,000,000 / 44,100 Hz = 23 microseconds for CD-quality.

The computer's sample period of 48 microseconds can be changed – see later. Under normal circumstances eight bits (one byte) represents amplitude. Think of a speaker cone vibrating in and out.

Many sound samples on the internet are in WAV format, filetype &fb1. To use these samples with WaveSynth they need to first be converted signed, mono, 8-bit format known as Armadeus, file type &d3c.

The sample period is stored in a 1-byte header before the wave form itself. Armadeus data then has to be converted to 'logarithmic' format. or 'log' for short, used by the RISC OS sound system. Listing 1 is single tasking but it will do the job.

Listing 1 Wav2d3e

```
REM WAV to Armadeus
REM (c) Drag N Drop Winter 2024
ON ERROR CLOSE#0:PRINT REPORT$+" at ";E
```

RockPro64

I was first introduced to the Rock Pro64 single board computer while watching Christopher Barnett's Youtube channel "Explaining Computers". The RockPro64 is a powerful single board computer from Pine64 launched in 2018 with the designation of LTS (Long Term Supply) and can still be purchased in 2024.

The ROCKPro64, features a Rockchip RK3399 hexa-core System on Chip (SoC) clocked at 1.8 GHz as well as a quad-core Mali-T860 MP4 GPU.

With a footprint of 125 × 80 × 19mm the board is much larger than a Raspberry Pi (85× 56×17mm). It's available with either 2 or 4 GB of memory, HDMI output supporting up to 4K, two USB 2 ports plus, one USB 3 port, one USB C port, a Pi-compatible 40 pin GPIO bus and eMMC module support up to 128GB.

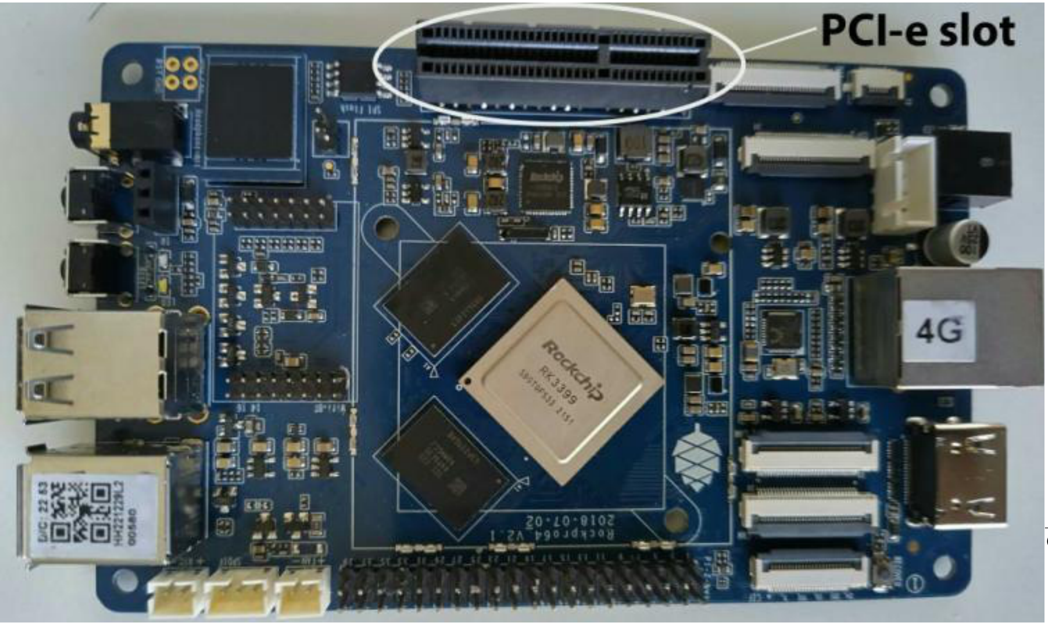
Best of all, it offers expansion possibilities through its' PCI-e 4×1 slot.

For anyone wanting to build a RAID storage system, Pine64 offer an aluminium NAS Case and a PCI-e to SATA adapter that supports two SATA drives.

Unusally, the board requires 12 volts power supply in 3 or 5 amps. The 5 amp supply would be required for anyone building a RAID setup.

Additionally Pine64 can supply a PCI-e to m.2 NVMe adapter that can accommodate NVMe drives ranging from 2230 to 2280.

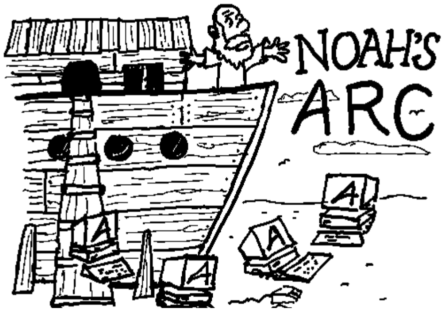
Figure 1. RockPro64 Single Board Computer (125×80mm)



Linux

Early in 2023, I purchased a 4GB board, a Pine64 preminium aluminium case, power supply and real time clock battery holder. The RockPro64 board is fitted upside down in the aluminium housing, so that the RK3399 chip is in contact with the heat sink pillar, contact

Noah's Arc



In this series we'll be celebrating software from RISC OS's precessor, the Archimedes, fondly called the Arc.

The Arc was the world's first dek-stop micro to use an ARM chip, a tiny thing which now powers millions of mobile phones and we don't give it a second thought.

At the time, the Arc was world beat-ing and it took years for the world to catch up – some word argue it hasn't caught up!

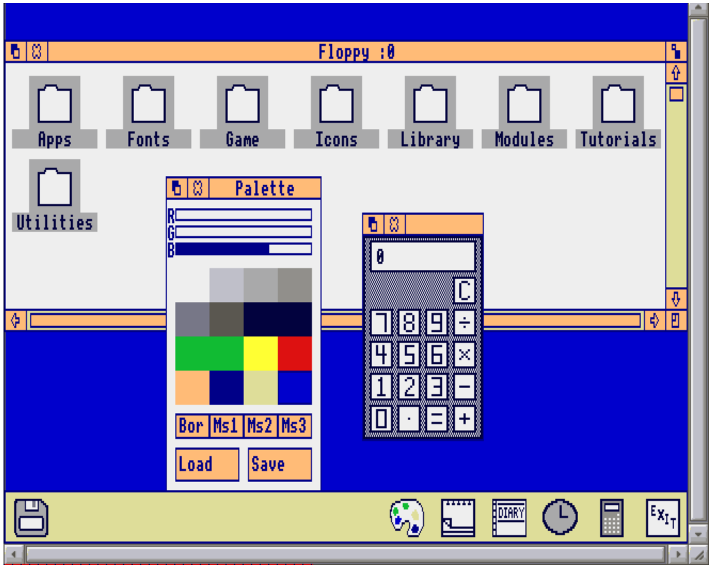
It ain't all nostalgia, however. There's a serious side to this. We'll be learning how to update legacy software to run properly on RISC OS Pi. You could just try to run this old software in emulators like ArchiEmu and ADFFS but that isn't always satisfactory.

Many excellent applications began their lives on the Arc, some fortunately survived to be modernised, others were left behind so we'll be carrying out a sort of rescue operation, mainly with BBC Basic software but also assembly code for updating relocatable modules.

We start by revisiting "Arc Windows Without Pain" from the March 1988 edition of *Acorn User*. This will be followed next time by "Go with the Flow" which develops a flow charting application, it ap-peared in Sep-tember 1988 edition of the mag.

Among the pages of this issue of *Drag 'N Drop* are digitally re-mastered versions of the articles. We couldn't possibly charge you for this content.

So it's all free, even with the sample edition of *Drag 'N Drop*, although our tutorial on updating the code is for full-edition, paying readers only.



The Archimedes desktop under emulation - reactions vary

RISC OZ

G'day mates, I hope you had a fantabulous Chrimbo and are ready for 2024. Firstly, big mobs to the boys over there at Drag N Drop for doing an ace job. I only wish I could come to one of your RISC OS shows. Maybe one day.

The whole continent of Straya desires to produce a window containing an icon that displays white coloured text in an outline font positioned on an orange background.

The text will be centered within the icon. To help you on your way, listing 1 starts up with icon flags set within procedure PROCICONS looks like this. As we go along, amend the F = &00000000 line to the value of F given and re-run the program.

Step 1: Create an icon block with all bytes containing a zero state:

0	0	0	0	0	0	0	0

F = &00000000 (=&0), nothing on screen.

Step 2: Orange is colour fourteen in decimal. Decimal 14 is E in hexadecimal. In hex, E is (1 x 23) + (1 × 22) + (1 × 21) + (0 × 20) = 8 + 4 +2 + 0 = 14 because 8 + 4 + 2 + 0 is 1000 + 100 + 10 + 0 in binary which in turn is 1110.

Bit 31 (nybble 7) of the icon flag block flag has hex value E and the icon flag is now &E0000000

E	0	0	0	0	0	0	0

F = &E0000000.

Nothing on screen

Listing 1

```
 REM SetFlagS - RISC OZ
 REM (c) Drag N Drop Winter 2024
 SYS "Wimp_Initialise",200,&4B534154,"He
llo world"
 DIM B 256, T 256, U 512, 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 A text icon,200,200,600,300
 DATA X,Y,X+W,Y+H,0,0,-1
 DATA &86001012           : REM window f
lags
 DATA &01000207,&000C0103 : REM window c
olours
 DATA 0,0,W,H
 DATA &00000109,&3000     : REM title ba
r and work flags
 DATA 1,0,T,0,0,0
```

ARC WINDOWS WITHOUT PAIN

The Arc's huge potential for using windows, icons, menus and pointers can be accessed simply from Basic

Chris Adie

One of the most exciting developments incorporated in the Archimedes micros has been the 'WIMP' system – the use of windows, icons, menus and pointers to represent a simulated 'desktop'. The Apple Macintosh has been the major force in popularising this way of working, and the MS-DOS world has followed Apple's lead, with the GEM and MS Windows environments. The Archimedes Desktop program on the Welcome disc shows that Acorn is also following the trend towards easier-to-use software.

What you may not have realised is that the Desktop is written in Basic! All the window handling, the pop-up menus and so forth are *controlled* from Basic, but actually *performed* by part of the Arthur operating system called the Window Manager. This division of work between the program and the operating system means that it is easy to write your own programs which use win- dows – much easier than in GEM, MS Windows or on the Macintosh! This article explains how your software can take advantage of this state-of-the-art user interface.

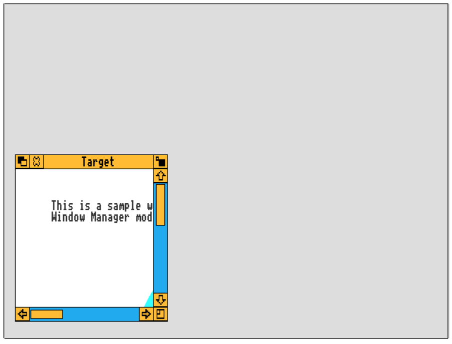
The SYS statement

We'll make a lot of use of the Basic SYS statement in this article. SYS was introduced in Basic 5 on the Archimedes as a way of calling the operating system. Most of the familiar BBC operating system routines, Osbyte, Oswrch and so on, can be called using SYS, although they can also be called in the old way too. SYS is a more sophisticated version of CALL.

When Basic finds a statement like

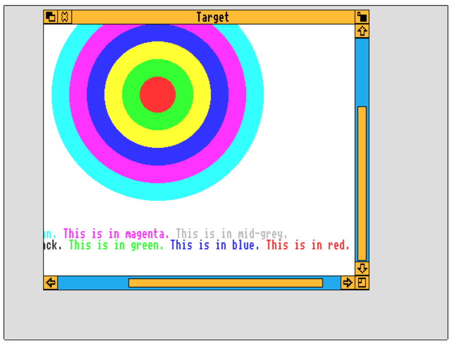
```
   10 SYS 6,A%,X%,Y% TO
 result%
```

it will copy A% to processor register R0, X% to register Rl, Y% to register R2, and then execute a software interrupt (SWI) instruction, type 6.



This is a sample w
Window Manager mod

Listing 1 produces a simple window



n. This is in magenta. This is in mid-grey.
ck. This is in green. This is in blue. This is in red.

It can be moved and resized

An SWI is an ARM machine code instruction which causes control to be transferred directly to the operating system. What happens then depends on the type of the SWI. Type 6 is equivalent to Osbyte, so the operating system performs the appropriate operation and then returns to the machine code instruction following the SWI. The Basic interpreter stores the value of RO returned by the operating system in the variable result%, and goes on to the next statement inthe program.

There are over 16 million possible SWI types, but only a very few of these are defined. The Window Manager uses 24 SWI types, starting with type &400C0 but we'll be dealing with only a few of them.

One problem with SYS is that only the first eight of the ARM processor's registers can be accessed. When you have more than eight 32-bit words of information to pass across, you must use a parameter block – an area of memory pointed to be one register, usually R1, which contains all the data for the operating system routine, and where the routine will store any output values before returning. So for example you could have

```
    DIM black &300
    SYS poll,0,block TO r
  eason%
```

where 'poll' is one of the Window Manager SWI types.

A single window

Listing 1 on the yellow pages is a short program which creates a single window on a mode 12 screen. It's important to realise we are talking about scrollable 'desktop' windows here, not the usual graphics and text windows defined with VDU 24 and VDU 28. When we need to talk about the VDU 24 graphics window, we'll call it the 'clip rectangle', since any graphics performed outside that region don't appear on the screen. The example window drawn by the program contains concentric coloured rings together with some text, and it has a title bar, full box, horizontal and vertical scroll bars and a size box. You can move the window around the screen, scroll and re-size it, and when you click on the close box the program terminates. We'll go through the program step-by-step so you can see what all the routines do.

With Arthur version 0.2, not all of the characters above ASCII 127 are pre-programmed with the shapes required by the Window Manager. If you still have version 0.2 then you'll have to add the

PROCdefinewimpcharacters. PROCsysvars defines the various Window Manager SWI types.

Before you can make any Window Manager calls, you must initialise the WIMP system with SYS init. This statement also has the effect of VDU 5, and all the text output to windows must be done in graphics mode. After that, there are two stages in preparing a window for display on the screen – first you create it, then you open it.

Creating a window is done with

```
  SYS createw,0,block T
O handle%
```

where createw=&400C1. From now on, I'll not refer explicitly to the SWI numbers - you can get them from PROCsysvars in the listing. The creation routine returns an integer called the window handle, which uniquely identifies the window. It is used in most other Window Manager calls. In many ways it's like a familiar file handle. The parameters block passed to createw contains quite a lot of information about the window. It controls features like the size and colour of the window - all the parameters are shown in tables 1 and 2.

Physical work area

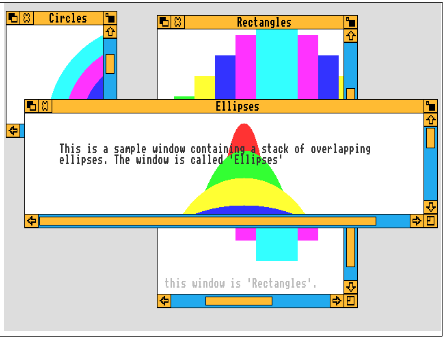
The physical work area (PWA) of a window is the part of the window containing information – it excludes the scroll bars and title bars. You can specify the position of the window on the screen by giving the co-ordinates of the lower left and upper right corners of the PWA, relative to the normal graphics origin at the bottom left-hand corner of the Archimedes screen as shown in figure 1.

Usually, the physical work area displays a small part of a larger work area, and the PWA can be moved around this larger logical work area (LWA) using the scroll bars and arrow icons. Figure 2 shows how the LWA and PWA relate to each other. We specify the size of the LWA in the parameter block for createw by giving the minimum and maximum X and Y values measured from point called the LWA origin. This must be in the LWA. In the Desktop program the LWA origin is at the top left corner of the LWA. This has the disadvantage that Y values within the LWA measured relative to the LWA origin are negative. So in the yellow pages programs the LWA origin is at the bottom left corner of the LWA, at the graphics origin. The LWA min X and LWA min Y arc both set to zero.

The positions of the scroll bars control

block!0 to block!12	Initial co-ordinate of window's physical work area (where first appears on the screen)
block!0	PWA min X
block!4	PWAmin Y
block!8	PWA max X
block!12	PWA max Y
block!16	Initial X scroll bar offset
block!20	Initial Y scroll bar offset
block!24	Handle of window under which to create this one (-1 if on top)
block!28	Window flags (see Table 2)
block?32	Window title foreground colour
block?33	Window title background colour
block?34	Work area foreground colour
block?35	Work area background colour
block?36	Scroll bar background colour
block?37	Scroll bar foreground colour
block?38	Title bar highlight colour
block?39	Reserved by Acorn
block!40 block!52	to Logical work area co-ordinates
block!40	LWA minX
block!44	LWA min Y
block!48	LWA max X
block!52	LWA max Y
block!56	Icon flags for title bar
block!69 to block!68	Reserved by Acorn
$(block+72)	Title string (up to 11 characters,terminated by CR)
block!84	Number of icons in this window
block!88 onwards	Icon data is stored here

Table1: Parameter block for createw



This is a sample window containing a stack of overlapping ellipses. The window is called 'Ellipses'

this window is 'Rectangles'.

Listing 2 gives you three windows

which part of the LWA appears in the PWA. The scroll bar positions are given by two numbers which are the co-ordinates of the top left of the PWA, relative to the LWA origin – this may explain why the Desktop program puts the LWA origin where it does!

Opening a window is easily done by calling the Window Manager's openw routine with the SYS statement:

```
    SYS openw,0,block
```

where the parameter block is set up as shown in table 3. If you don't know what values to fill the block with, you can use:

```
   block! = handlez%
   SYS getw,0,block
```

to set the block up ready for openw. This opens the window in its last known position on the screen.

WIMP events

The biggest difference between a program using windows and a conventional program is that in the first case, the user is in charge of what happens next, not the programmer. You can move, scroll and resize windows, click the mouse on objects in the windows, or make keystrokes. The program must know what to do in all these circumstances, though it may of course choose to ignore some of them. The main loop at the heart of a program using the Window Manager reflects this situation:

```
REPEAT
    Wait for the user to
        do something
    Take appropriate action
UNTIL finished
```

To find out what the user has done, you use a routine called poll. The only input in the parameter block to poll is a 'mask' indicating the actions you don't want the user to be able to do – if it is 0, everything is allowable. On leaving poll, register RO contains the kind of action which took place, and the parameter block carries information specific to that action. These actions are called 'events', but they are not the same as 'events' on the BBC micro - those can only be used from machine code.

There are 10 possible values returned by poll, as shown in table 4. The information returned in the parameter block for each is shown in table 5.

Event 0 is easy – the user hasn't done anything, and we don't need to do anything either, unless you have an animated display to take care of, like the clock in the Desktop for instance.

Event 2 (open window request) informs us that the user has brought a window to the top of the stack of windows by clicking on the title bar, or that the window's size has changed or scrolled. Usually, a sufficient response to event 2 is to call openw with the parameter block passed back by the poll.

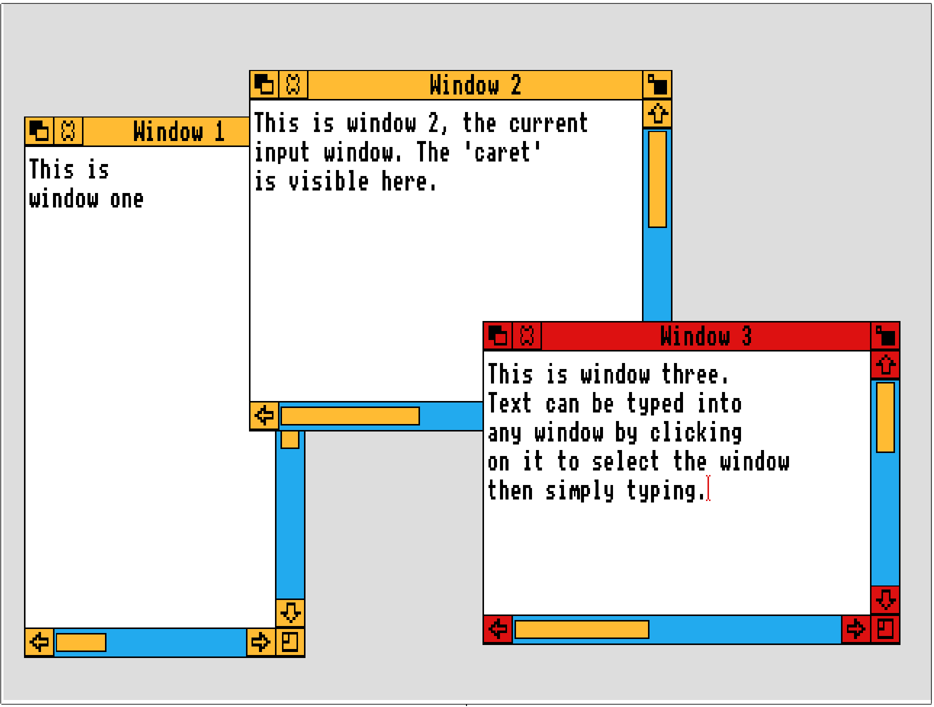
The openw routine by itself does not result in anything actually being drawn on the screen. A call to openw will usually result in all or part of the window and its borders needing to be redrawn, so openw generates event 1 (redraw window request) before it returns control to your program. You must respond to this event 1 in a very definite way, using the following code:

```
   SYS redraww,0,block T
O more%
   WHILE more%
       PROCredrawwindow
       SYS getr,0,block
TO more%
   ENDWHILE
```

The routine redraww works out a list of invalid rectangles, which cover the visible parts of the window which need updating. It then redraws the title bar and scroll bar where these intersect with the list, and finally it indicates to your program whether part of the PWA needs updating by returning a flag in register R0. The parameter block returned by redraww is shown in table 6. The VDU 24 graphics clip rectangle is set to the co-ordinates at block!28 to block!40, and your PROCredrawwindow routine takes responsibility for drawing the appropriate graphic or text in that rectangle. The process is repeated for all rectangles in the invalid list, using routine getr to get position parameters for each in turn, until the more% flag returned in RO becomes FALSE. Note that more than one window's PWA may need updating, so PROCredrawwindow needs to pay attention to the window handle returned in block!0.

The above sounds an incredibly complicated procedure for doing something which is in reality quite easy to understand. A couple of examples will probably make things clearer. Diagram 3 shows what happens when a window which is overlapped by two other windows is brought to the top, by clicking on the title bar. First of all, you get event 2 (open window request) from poll, so we call openw. We then get event 1 (redraw window request), and so we call redraww, which draws the parts of the border which were hidden by the other windows. Redraww passes back the co-ordinates of invalid rectangle A, which you draw in, and then we call getr. This passes back the co-ordinates of rectangle B, so we draw in there too, and call getr again. But this time getr returns a more% value of FALSE showing there are no more rectangles. So you go back to calling poll. Notice that redraww clears all the rectang- les to the window background colour, ready for you to draw on.

Diagram 4 shows what happen when you scroll a window which is partially overlapped by another window. The poll routine returns an open window request event, and you call openw. Poll then gives you a redraw window request event.



This is
window one

This is window 2, the current
input window. The 'caret'
is visible here.

This is window three.
Text can be typed into
any window by clicking
on it to select the window
then simply typing.

Text can be added with the caret

Bit	Meaning if bit set
0	Window has a title bar
1	Window can be moved
2	There is a vertical scroll bar
3	There is a horizontal scroll bar
4	The window can be redrawn by Window Manager (ie, it contains icons only)
5	The window is a 'pane' in another window
6	The window is allowed to go outside the screen
7	There are no 'send-to-back' or 'close' boxes

The following bits are output flags, set by certain Window Manager routines.

16	The window is open
17	The window is not covered by another
18	The window has been toggled to full size

Table 2: Window flags

block!0	Handle of window
block!4 to block!16	Co-ordinates of Physical work area
block!4	PWA min X
block!8	PWA min Y
block!12	PWA max X
block!16	PWA max Y
block!20	X scroll bar offset
block!24	Y scroll bar offset
block!28	Handle of window which this one is behind (-1 if it's on top)

Table 3: Parameter block for openw and getw

Event code	Meaning
0	Nothing has happened
1	A window needs redrawing
2	A window needs (re)opening
3	A window close box has been clicked on
4	The pointer is leaving a window
5	The pointer is entering a window
6	User has pressed or released a mouse button
7	User has finished a drag box operation (after SWI drag)
8	User has pressed a key
9	User has selected a pop-up menu item

Table 4: User event codes returned by poll

User event	1	2	3	4	5
block! 0	Handle	Handle	Handle	Handle	Handle
block! 4		PWaminX			
block! 8		PWaminY			
block!12		PWamaxX			
block!16		PWamaxY			
block!20		SerX			
block !24		ScrY			
block !28		Under			

User event	6	7	8	9
block!0	MouseX	DrpminX	Handle	Mainitem
block!4	MouseY	DrgminY	Icnhandle	Subitem
block!8	Newbut	DrgmaxX	CaretX	
block!2	Handle	DrgmaxY	CaretY	
block!16	Icnhandle		Key	
block!20	Oldbut			
blocki24				
block !28				

Table 5: Parameter block data returned by poll

You call redraww, which draws the scroll bars in their new position and copies those parts of the physical work area which will

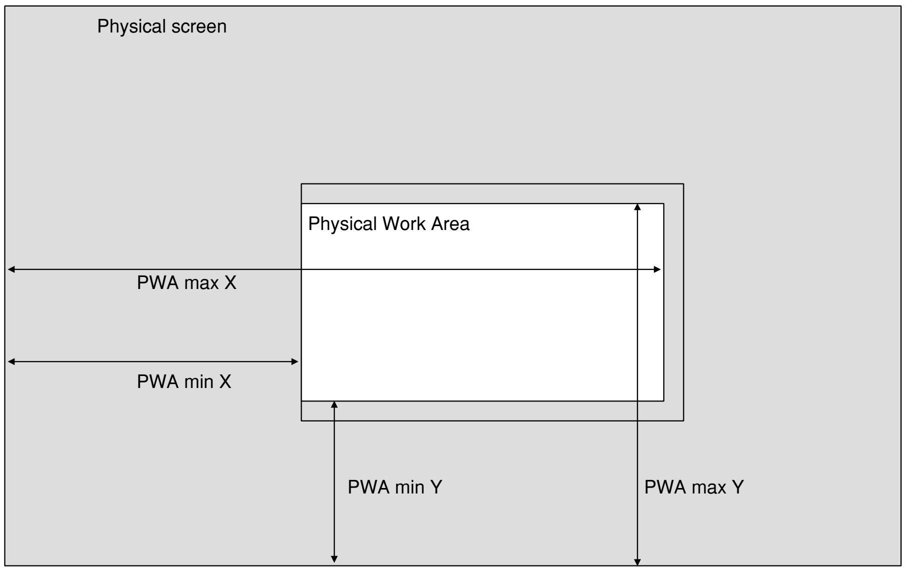


Diagram 1: Relationship of physical work area to the Archimedes' screen

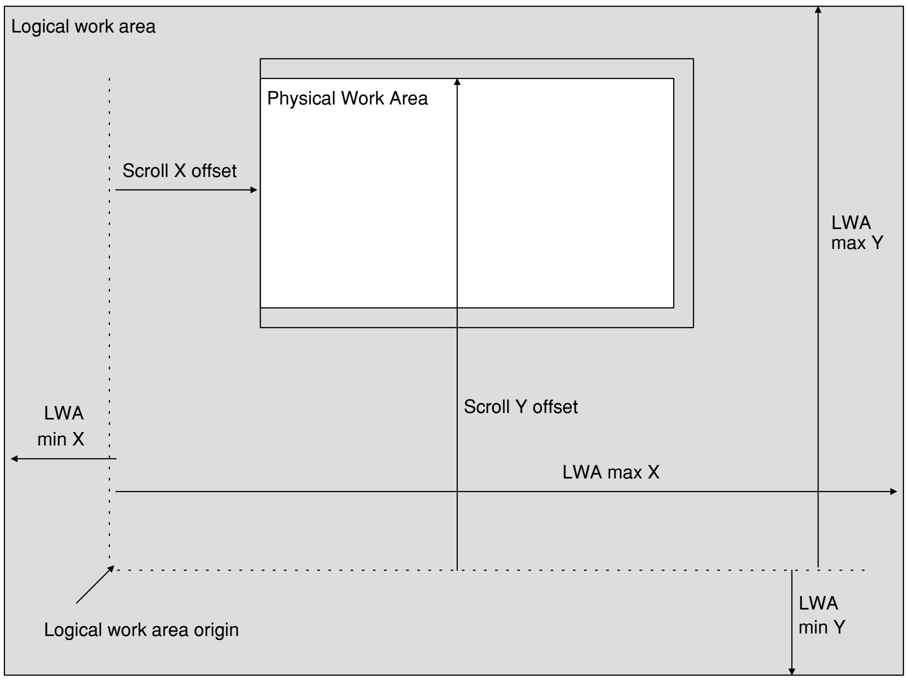


Diagram 2: How the logical and physical work areas relate

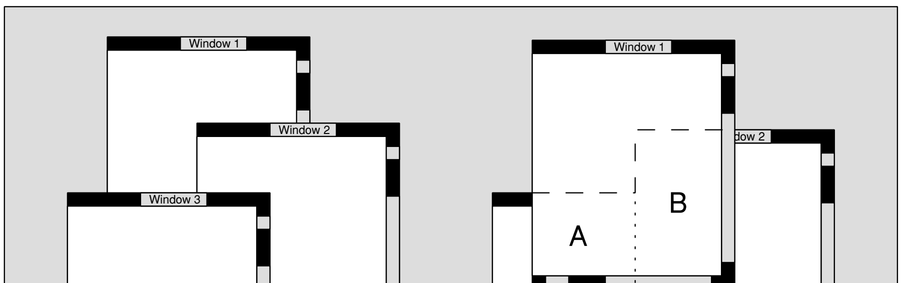


Diagram 3: On the left, window 1 lies underneath, so when it is brought to the front (right), rectangles A and B need redrawing

remain visible to their new location. It returns the co-ordinates of rectangle A, which you then draw in and call getr for the co-ordinates of rectangle B. Having drawn that in, getre indicates no more rectangles, so you can go back to calling poll. You can see that in this case you don't need to redraw any part of the house, because it does not appear in any of the invalid rectangles.

If you like, try to work out what happens when you go from the situation in diagram 3 – in other words when the user clicks on the 'send to back' box in window 1. Remember that events tell you which window they are concerned with, by including the handle in the parameter block information returned by getr.

When the user clicks on the close box of a window, event 3 (close window request) is passed back by poll. You call the closew routine with the appropriate window handle in block!0. If necessary you will get open window and redraw window requests when you go back to calling poll, for windows revealed from underneath the closed one.

Text and graphics

PROCredrawwindow and PROCdraw in listing 1 show how to put text and graphics in a window. In fact, for this simple program, the entire LWA is redrawn for each rectangle that we are passed by redraww and getr, but since the graphics clip rectangle is set, only the rectangle in question is actually updated.

The first thing to do is to find out where the LWA origin is, relative to the graphics origin at the bottom-left corner of the screen. Diagrams 1 and 2 show that you can calculate this from the PWA co-ordinates and the scroll offsets. All text and graphics to be drawn in the window must be drawn relative to the LWA origin, so that scrolling will have the correct effect. The obvious way to do this is to move the screen graphics origin to the LWA origin while we do the drawing, and move it back afterwards, but unfortunately this leads to problems. In PROCdraw in listing 1, you must add the LWA origin co-ordinates to every graphics co-ordinate in the LWA.

Remember that VDU 5 mode is always in effect, so that text as well as graphics is limited to the graphics clip rectangle. You must position the graphics cursor using MOVE before using PRINT, and you can't use Return for a new linc, because that will move the graphics cursor to the left-hand edge of the current clip rectangle, not at all what you want.

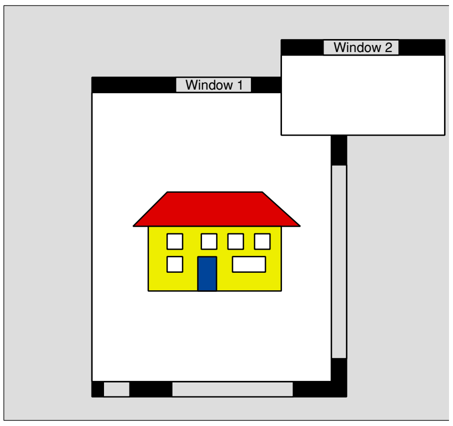


Diagram 4: Scrolling a partially obscured window

Multiple windows

Having typed in listing 1, relatively few changes are required to get to listing 2. This displays three windows containing circles, ellipses and rectangles. You simply have an array, window%(), to hold the window handles, and another array, windows$(), to hold the window titles. PROCredrawwindow works out the name of the window to redraw and calls FNdrawcircles, FNdrawrectangles or FNdrawellipses appropriately. You can increase the number of windows (controlled by 'maxindex') up to 32, so long as you prove a suitable drawing routine for each. The structure of the program is quite independent of whether you are using two or 32 windows.

Text entry

The Desktop program has a 'notebook' window, which you can type into. Typed characters appear at the position of a text cursor, shaped rather like an elongated capital I, This is called a 'caret', and there are two special routines in the Window Manager for dealing with it.

The first of these routines is setcarct. It doesn't use a parameter block. Instead, all the information is passed in registers. The call to setcaret is typically like this:

```
  SYS setcaret,handle,
-1,X,Y,&1000024,-1
```

where X and Y give the required position of the caret relative to the window's LWA origin. The three parameters which are given explicit numeric values have to do with the size of the caret and whether it is associated with a particular icon.

A call to setcaret results in the window title bar changing colour, to that specified in block?38 in the original call to createw, and the caret being drawn in the indicated position (providing that the position is within the visible PWA).

block!0	Handle of window
block!4 to block!16	Co-ordinates of physical work area
block!4	PWA min X
block!8	PWA min Y
block!12	PWA max X
block!16	PWA max Y
block!20	X scroll bar offset
block!24	Y scroll bar offset
block!28 to block!40	Co-ordinates of rectangle to draw
block!28	min X
block!32	min Y
block !36	max X
block!40	max Y

Table 6: Block from redraww, getr and updatew

JARGON BOX

Window - A discrete area of the screen used to display information. There may be several on the screen at once.
Icon – Small picture on the screen which causes some action when the mouse button is clicked on it.
Close Box - Clicking on this icon removes a window from the sereen.
Full Box – An icon which enlarges the window to full screen size.
Scroll Bar – Device on the side of the window to let you change the view through the window.

The window now has the 'input focus' – in other words a key pressed event (poll event 8) will pass back the handle of this window and the carct position in the parameter block, A setcaret call to another window will erase the caret from the window with the current input focus and reset the title bar back to normal before putting the caret in the newly selected window.

Listing 3 is a program which displays three windows. You can click the move ina window work area and type in your own text, rather like the Desktop notebook. [f you've already typed in listing 2, the changes you'll need to make are minor. The text for all the windows is stored as a two-dimensional array of 80-character strings. In PROCwritetext, which writes out the text when a redraw window request is received, this program actually pays attention to the rectangle it's being asked to redraw, and it only writes out those charac- ters which appear in that rectangle. This speeds things up enormously compared to writing out the entire text every time.

When a key pressed event is detected by poll, PROCkeypressed places the character in the text array, and works out where in the LWA the character should be displayed. It calls frerdrw, and then moves the caret one position along. The frerdrw (force redraw) routine is called with:

```
      SYS frcrdrw,handle,x1
  ,y1,x2,y2
```

where x1, v1, x2, y2 are the co-ordinates, relative to the LWA origin, of the rectangle which needs to be redrawn. The next call to poll will result in a redraw window request event if any part of that rectangle is visible on the sercen.

The other routine for handling the caret is getcaret. Although it's not actually used in listing 3, it can be called with:

```
  SYS getcaret,0,block
```

On return, the parameter block contains the information which was passed in the fast setcaret call, so in theory your program does not even need to keep track of where the caret is on the screen!

Icons and menus

We've mentioned both icons and menus in passing, but haven't said much about them. A future article will cover these aspects of the WIMP environment, but for now you have got enough information and examples to program your own scrollable windows. Experiment with the existing programs by increasing the number of windows or putting different things in them. Try altering listing 3 so chat the caret is positioned wherever you click the mouse. There are all sorts of possibilities which scrolling windows make available - consider what benefits would a really large logical work area give you, for instance.

Start thinking now about how you could alter your own programs to take advantage of scolling windows. Several of Acorn Us- er's own listings could be radically im- proved by a combination of windows and mouse control.

Over the coming months we'll take a look at how to write Arc menus and use the mouse in your programming.

The listings from this article are on the yellow pages, see page 113. They are also available on Arc-format discs – see page 13.

Listing 1. Single window demo

```
   10 REM Windows Demo 1
   20 REM by Chris Adie
   30 REM for Arc only
   40 REM (C) Acorn User March 1988
   50 :
   60 MODE 12
   70 ON ERROR PROCerror
   80 *FX 4,1
   90 *FX 200,1
  100 PROCsysvars
  110 PROCcolours
  120 PROCdefinewimpcharacters
  130 DIM block &300
  140 SYS init
  150 SYS frcrdrw,-1,0,0,1279,1023
  160 *POINTER
  170 window%=FNcreatewindow("Target",&0
F,wblack,wwhite,1279,1023,10,10,400,400)
  180 PROCopenwindow(window%,FALSE)
  190 finished%=FALSE
  200 :
  210 REPEAT
  220 event%=FNpoll(0)
  230 PROCaction(event%,finished%)
  240 UNTIL finished%
  250 :
  260 *FX 4,0
  270 *FX 200,0
  280 MODE 12
  290 END
  300 :
  310 DEF FNpoll(mask%)
  320 LOCAL a%
  330 block!0=mask%
  340 SYS poll,0,block TO a%
  350 =a%
  360 :
  370 DEF PROCaction(evnt%,RETURN end%)
  380 CASE evnt% OF
  390 WHEN 0 :
  400 WHEN 1 :
  410 PROCredrawwindow(block!0)
  420 WHEN 2 :
  430 PROCopenwindow(block!0,TRUE)
  440 WHEN 3 :
  450 PROCclosewindow(block!0)
  460 end%=TRUE
  470 WHEN 4 :
  480 WHEN 5 :
  490 WHEN 6 :
  500 WHEN 7 :
  510 WHEN 8 :
  520 WHEN 9 :
  530 ENDCASE
  540 ENDPROC
  550 :
  560 DEF PROCopenwindow(handle%,full%)
  570 block!0=handle%
  580 IF NOT full% THEN SYS getw,0,block
  590 SYS openw,0,block
  600 ENDPROC
  610 :
  620 DEF PROCredrawwindow(handle%)
  630 LOCAL more%,x0%,y0%
  640 block!0=handle%
  650 SYS redraww,0,block TO more%
  660 PROClwaorigin(block+4,x0%,y0%)
  670 WHILE more%
  680 PROCdraw(x0%,y0%)
  690 SYS getr,0,block TO more%
  700 ENDWHILE
  710 ENDPROC
  720 :
  730 DEF PROCclosewindow(handle%)
  740 block!0=handle%
  750 SYS closew,0,block
  760 ENDPROC
  770 :
  780 DEF PROClwaorigin(b,RETURN x%,RETU
RN y%)
  790 x%=b!0-b!16
  800 y%=b!12-b!20
  810 ENDPROC
  820 :
  830 DEF PROCdraw(x0%,y0%)
  840 LOCAL i%
  850 FOR i%=300 TO 50 STEP -50
  860 GCOL FNgcol(i% DIV 50)
  870 CIRCLE FILL x0%+640,y0%+512,i%
  880 NEXT
  890 GCOL black
  900 MOVE x0%+100,y0%+932
  910 PRINT "This is a sample window, sh
owing the scrolling features of the"
  920 MOVE x0%+100,y0%+900
  930 PRINT "Window Manager module in th
e Acorn Archimedes microcomputer."
  940 MOVE x0%+100,y0%+100
  950 PRINT "This is in Black. ";
  960 GCOL green:PRINT "This is in green
. ";
  970 GCOL blue:PRINT "This is in blue.
";
  980 GCOL red:PRINT "This is in red.";
  990 MOVE x0%+100,y0%+132
 1000 GCOL cyan:PRINT "This is in cyan.
";
 1010 GCOL magenta:PRINT "This is in mag
enta. ";
 1020 GCOL midgrey:PRINT "This is in mid
-grey. ";
 1030 ENDPROC
 1040 :
 1050 DEF PROCcolours
 1060 black=0
 1070 red=1
 1080 green=2
 1090 yellow=3
 1100 blue=4
 1110 magenta=5
 1120 cyan=6
 1130 white=7
 1140 midgrey=15
 1150 scrollbarfgcol=14
 1160 scrollbarbgcol=13
 1170 highlightbgcol=red
 1180 titlefgcol=12
 1190 titlebgcol=scrollbarfgcol
 1200 VDU 19,0,24,128,128,128
 1210 VDU 19,15,16,128,128,128
 1220 VDU 19,14,16,15*16,11*16,6*16
 1230 VDU 19,13,16,0*16,12*16,15*16
 1240 VDU 19,12,16,0*16,0*16,8*16
 1250 VDU 19,11,16|
 1260 VDU 19,10,16|
 1270 VDU 19,9,16|
 1280 VDU 19,8,16|
 1290 ENDPROC
 1300 :
 1310 DEF PROCsysvars
 1320 wimp=&400C0
 1330 init=wimp+0
 1340 createw=wimp+1
 1350 createi=wimp+2
 1360 deletew=wimp+3
 1370 deletei=wimp+4
 1380 openw=wimp+5
 1390 closew=wimp+6
 1400 poll=wimp+7
 1410 redraww=wimp+8
 1420 updatew=wimp+9
 1430 getr=wimp+10
 1440 getw=wimp+11
 1450 getwi=wimp+12
 1460 seti=wimp+13
 1470 geti=wimp+14
 1480 getp=wimp+15
 1490 drag=wimp+16
 1500 frcrdrw=wimp+17
 1510 setcaret=wimp+18
 1520 getcaret=wimp+19
 1530 createm=wimp+20
 1540 decodem=wimp+21
 1550 whichi=wimp+22
 1560 setextent=wimp+23
 1570 ENDPROC
 1580 :
 1590 DEF FNcreatewindow(title$,flags%,f
gcol%,bgcol%,maxx%,maxy%,wal%,wab%,war%,
wat%)
 1600 LOCAL handle%
 1610 block!0=wal%
 1620 block!4=wab%
 1630 block!8=war%
 1640 block!12=wat%
 1650 block!16=0
 1660 block!20=maxy%
 1670 block!24=-1
 1680 block!28=flags%
 1690 block?32=titlefgcol
 1700 block?33=titlebgcol
 1710 block?34=fgcol%
 1720 block?35=bgcol%
 1730 block?36=scrollbarbgcol
 1740 block?37=scrollbarfgcol
 1750 block?38=highlightbgcol
 1760 block?39=0
 1770 block!40=0
 1780 block!44=0
 1790 block!48=maxx%
 1800 block!52=maxy%
 1810 block!56=&2D
 1820 block!60=&3000 :block!64=0:block!6
8=0
 1830 $(block+72)=LEFT$(title$,11)
 1840 block!84=0
 1850 SYS createw,0,block TO handle%
 1860 =handle%
 1870 :
 1880 DEF PROCdefinewimpcharacters
 1890 REM lines 1900-2050 are not needed
 with Arthur > 0.2
 1900 VDU 23,128,102,0,60,102,126,102,10
2,0
 1910 VDU 23,129,240,144,240,31,31,31,31
,0
 1920 VDU 23,130,224,224,224,31,17,17,31
,0
 1930 VDU 23,131,254,146,146,242,130,130
,254,0
 1940 VDU 23,132,102,153,129,66,129,153,
102,0
 1950 VDU 23,133,252,252,255,225,225,33,
63,0
 1960 VDU 23,134,102,0,102,102,102,102,6
0,0
 1970 VDU 23,135,126,195,157,177,157,195
,126,0
 1980 VDU 23,136,24,40,79,129,79,40,24,0
 1990 VDU 23,137,24,20,242,129,242,20,24
,0
 2000 VDU 23,138,60,36,36,231,66,36,24,0
 2010 VDU 23,139,24,36,66,231,36,36,60,0
 2020 VDU 23,140,48,24,60,6,62,102,62,0
 2030 VDU 23,141,48,24,60,102,126,96,60,
0
 2040 VDU 23,142,102,0,60,102,126,96,60,
0
 2050 VDU 23,143,60,102,60,102,126,96,60
,0
 2060 ENDPROC
 2070 :
 2080 DEF PROCerror
 2090 MODE 12
 2100 *FX 4,0
 2110 *FX 200,0
 2120 $block="Error":SYS "Wimp_CommandWi
ndow",block
 2130 REPORT:PRINT " at line ";ERL
 2140 END
```

Listing 2. Multiple window demo

Listings 2 and 3 also require PROCcolours, PROCsysvars, FNcreatewindow, PROCdefinewimpcharacters and PROCerror from listing 1. Add these procedures onto the end of the listing

```
   10 REM Windows Demo 2
   20 REM by Chris Adie
   30 REM for Arc only
   40 REM (C) Acorn User March 1988
   50 :
   60 MODE 12
   70 ON ERROR PROCerror
   80 REM*FX 4,1
   90 REM*FX 200,1
  100 PROCsysvars
  110 PROCcolours
  120 REMPROCdefinewimpcharacters
  130 DIM block &300
  140 SYS init
  150 SYS frcrdrw,-1,0,0,1279,1023
  160 *POINTER
  170 maxindex=3
  180 DIM window%(maxindex)
  190 DIM window$(maxindex)
  200 DATA "Circles","Ellipses","Rectang
les"
  210 FOR i%=1 TO maxindex
  220 READ window$(i%)
  230 window%(i%)=FNcreatewindow(window$
(i%),&0F,wblack,wwhite,1279,1023,100*i%,
150+100*i%,400+100*i%,650+100*i%)
  240 PROCopenwindow(window%(i%),FALSE)
  250 NEXT i%
  260 :
  270 REPEAT
  280 PROCaction(FNpoll(0))
  290 UNTIL FNfinished
  300 :
  310 *FX 4,0
  320 *FX 200,0
  330 MODE 12
  340 END
  350 :
  360 DEF FNpoll(mask%)
  370 LOCAL a%
  380 block!0=mask%
  390 SYS poll,0,block TO a%
  400 =a%
  410 :
  420 DEF PROCaction(evnt%)
  430 CASE evnt% OF
  440 WHEN 0 :
  450 WHEN 1 :
  460 PROCredrawwindow(block!0)
  470 WHEN 2 :
  480 PROCopenwindow(block!0,TRUE)
  490 WHEN 3 :
  500 PROCclosewindow(block!0)
  510 window%(FNwhichwindow(block!0))=-1
  520 WHEN 4 :
  530 WHEN 5 :
  540 WHEN 6 :
  550 WHEN 7 :
  560 WHEN 8 :
  570 WHEN 9 :
  580 ENDCASE
  590 ENDPROC
  600 :
  610 DEF PROCopenwindow(handle%,full%)
  620 block!0=handle%
  630 IF NOT full% THEN SYS getw,0,block
  640 SYS openw,0,block
  650 ENDPROC
  660 :
  670 DEF PROCredrawwindow(handle%)
  680 LOCAL more%,x0%,y0%,i%,name$,junk
  690 i%=FNwhichwindow(handle%)
  700 name$=window$(i%)
  710 block!0=handle%
  720 SYS redraww,0,block TO more%
  730 PROClwaorigin(block+4,x0%,y0%)
  740 WHILE more%
  750 junk=EVAL("FNdraw"+name$+"(x0%,y0%
)")
  760 SYS getr,0,block TO more%
  770 ENDWHILE
  780 ENDPROC
  790 :
  800 DEF PROCclosewindow(handle%)
  810 block!0=handle%
  820 SYS closew,0,block
  830 ENDPROC
  840 :
  850 DEF PROClwaorigin(b,RETURN x%,RETU
RN y%)
  860 x%=b!0-b!16
  870 y%=b!12-b!20
  880 ENDPROC
  890 :
  900 DEF FNdrawCircles(x0%,y0%)
  910 LOCAL i%,j%
  920 FOR i%=300 TO 50 STEP -50
  930 GCOL FNgcol(i% DIV50)
  940 CIRCLE FILL x0%+640,y0%+512,i%
  950 NEXT
  960 GCOL black
  970 MOVE x0%+100,y0%+932
  980 PRINT "This is a sample window, sh
owing the scrolling features of the"
  990 MOVE x0%+100,y0%+900
 1000 PRINT "Window Manager module in th
e Acorn Archimedes microcomputer."
 1010 MOVE x0%+100,y0%+100
 1020 PRINT "This is in Black. ";
 1030 GCOL green:PRINT "This is in green
. ";
 1040 GCOL blue:PRINT "This is in blue.
";
 1050 GCOL red:PRINT "This is in red.";
 1060 MOVE x0%+100,y0%+132
 1070 GCOL cyan:PRINT "This is in cyan.
";
 1080 GCOL magenta:PRINT "This is in mag
enta. ";
 1090 GCOL midgrey:PRINT "This is in mid
-grey. ";
 1100 =0
 1110 :
 1120 DEF FNdrawEllipses(x0%,y0%)
 1130 LOCAL i%,j%
 1140 FOR i%=1 TO 6
 1150 GCOL FNgcol(i%) :REM GCOL i%
 1160 ELLIPSE FILL x0%+640,y0%+512,80*i%
,80*(7-i%)
 1170 NEXT i%
 1180 GCOL black
 1190 MOVE x0%+100,y0%+932
 1200 PRINT "This is a sample window con
taining a stack of overlapping";
 1210 MOVE x0%+100,y0%+900
 1220 PRINT "ellipses. The window is cal
led 'Ellipses'"
 1230 GCOL magenta
 1240 MOVE x0%+100,y0%+132
 1250 PRINT "Try changing routine 'FNdra
wEllipses' to do something different."
 1260 =0
 1270 :
 1280 DEF FNdrawRectangles(x0%,y0%)
 1290 LOCAL i%,j%
 1300 FOR i%=1 TO 6
 1310 GCOL FNgcol(i%)
 1320 RECTANGLE FILL x0%+200+i%*60,y0%+5
12-i%*60,120*(7-i%),120*i%
 1330 NEXT i%
 1340 GCOL black
 1350 MOVE x0%+100,y0%+900
 1360 PRINT "Sample window containing a
stack of overlapping rectangles."
 1370 GCOL midgrey
 1380 MOVE x0%+100,y0%+100
 1390 PRINT "The name of this window is
'Rectangles'."
 1400 =0
 1410 :
 1420 DEF FNfinished
 1430 LOCAL finished,i%
 1440 finished=TRUE
 1450 FOR i%=1 TO maxindex
 1460 IF window%(i%)<>-1 THEN finished=F
ALSE
 1470 NEXT i%
 1480 =finished
 1490 :
 1500 DEF FNwhichwindow(handle%)
 1510 LOCAL i%,this%
 1520 i%=1:this%=-1
 1530 FOR i%=1 TO maxindex
 1540 IF window%(i%)=handle% THEN this%=
i%
 1550 NEXT
 1560 =this%
```

Listing 3. Notepad window demo

```
   10 REM Windows Demo 3
   20 REM by Chris Adie
   30 REM for Arc only
   40 REM (C) Acorn User March 1988
   50 :
   60 MODE 12
   70 ON ERROR PROCerror
   80 *FX 4,1
   90 *FX 200,1
  100 PROCsysvars
  110 PROCcolours
  120 PROCdefinewimpcharacters
  130 DIM block &300
  140 SYS init
  150 REMSYS frcrdrw,-1,0,0,1279,1023
  160 REM*POINTER
  170 maxindex=3
  180 DIM window%(maxindex)
  190 DIM window$(maxindex)
  200 DIM text$(maxindex,25)
  210 DIM xpos%(maxindex)
  220 DIM ypos%(maxindex)
  230 chx%=16
  240 chy%=40
  250 :
  260 DATA "Window 1","Window 2","Window
 3"
  270 FOR i%=1 TO maxindex
  280 READ window$(i%)
  290 FOR j%=1 TO 25
  300 text$(i%,j%)=STRING$(80," ")
  310 NEXT j%
  320 xpos%(i%)=1
  330 ypos%(i%)=1
  340 window%(i%)=FNcreatewindow(window$
(i%),&0F,wblack,wwhite,1279,1023,100*i%,
150+100*i%,400+100*i%,650+100*i%)
  350 PROCopenwindow(window%(i%),FALSE)
  360 NEXT i%
  370 :
  380 REPEAT
  390 PROCaction(FNpoll(0))
  400 UNTIL FNfinished
  410 :
  420 *FX 4,0
  430 *FX 200,0
  440 MODE 12
  450 END
  460 :
  470 DEF FNpoll(mask%)
  480 LOCAL a%
  490 block!0=mask%
  500 SYS poll,0,block TO a%
  510 =a%
  520 :
  530 DEF PROCaction(evnt%)
  540 CASE evnt% OF
  550 WHEN 0 :
  560 WHEN 1 :
  570 PROCredrawwindow(block!0)
  580 WHEN 2 :
  590 PROCopenwindow(block!0,TRUE)
  600 WHEN 3 :
  610 PROCclosewindow(block!0)
  620 window%(FNwhichwindow(block!0))=-1
  630 WHEN 4 :
  640 WHEN 5 :
  650 WHEN 6 :
  660 PROCbuttonchange
  670 WHEN 7 :
  680 WHEN 8 :
  690 PROCkeypressed
  700 WHEN 9 :
  710 ENDCASE
  720 ENDPROC
  730 :
  740 DEF PROCopenwindow(handle%,full%)
  750 block!0=handle%
  760 IF NOT full% THEN SYS getw,0,block
  770 SYS openw,0,block
  780 ENDPROC
  790 :
  800 DEF PROCredrawwindow(handle%)
  810 LOCAL more%,x0%,y0%,i%
  820 i%=FNwhichwindow(handle%)
  830 block!0=handle%
  840 SYS redraww,0,block TO more%
  850 PROClwaorigin(block+4,x0%,y0%)
  860 WHILE more%
  870 PROCwritetext(x0%,y0%,i%)
  880 SYS getr,0,block TO more%
  890 ENDWHILE
  900 ENDPROC
  910 :
  920 DEF PROCclosewindow(handle%)
  930 block!0=handle%
  940 SYS closew,0,block
  950 ENDPROC
  960 :
  970 DEF PROClwaorigin(b,RETURN x%,RETU
RN y%)
  980 x%=b!0-b!16
  990 y%=b!12-b!20
 1000 ENDPROC
 1010 :
 1020 DEF PROCbuttonchange
 1030 LOCAL i%
 1040 i%=FNwhichwindow(block!12)
 1050 IF ((block!8 AND %100)<>0) AND (bl
ock!12<>-1) THEN PROCcursor(i%,xpos%(i%)
,ypos%(i%))
 1060 ENDPROC
 1070 :
 1080 DEF PROCkeypressed
 1090 LOCAL i%,col%,row%,x1%,y1%,x2%,y2%
 1100 i%=FNwhichwindow(block!0)
 1110 col%=xpos%(i%)
 1120 row%=ypos%(i%)
 1130 CASE block!24 OF
 1140 WHEN 13:
 1150 row%+=1
 1160 col%=1
 1170 WHEN 7,&18C:
 1180 col%-=1
 1190 WHEN &18D:
 1200 col%+=1
 1210 WHEN &18F:
 1220 row%-=1
 1230 WHEN &18E:
 1240 row%+=1
 1250 OTHERWISE:
 1260 block!24=block!24 AND &7F
 1270 MID$(text$(i%,row%),col%,1)=CHR$(b
lock!24)
 1280 x1%=4+chx%*(col%-1)
 1290 y1%=1024-12-chy%*row%
 1300 x2%=x1%+chx%
 1310 y2%=y1%+chy%
 1320 SYS frcrdrw,block!0,x1%,y1%,x2%,y2
%
 1330 col%+=1
 1340 ENDCASE
 1350 IF col%>80 THEN col%=1:row%+=1
 1360 IF col%<1 THEN col%=80:row%-=1
 1370 IF (row%>25)OR(row%<1) THEN ENDPRO
C
 1380 PROCcursor(i%,col%,row%)
 1390 xpos%(i%)=col%
 1400 ypos%(i%)=row%
 1410 ENDPROC
 1420 :
 1430 DEF PROCwritetext(x0%,y0%,index%)
 1440 LOCAL i%,a%,b%,c%,d%,x1%,y1%,x2%,y
2%
 1450 PROClwacliprectangle(block+4,x1%,y
1%,x2%,y2%)
 1460 a%=(1024-12-y2%) DIV chy%+1
 1470 b%=(1024-12-y1%) DIV chy%+2
 1480 c%=(x1%-4) DIV chx%+1
 1490 d%=(x2%-4) DIV chx%+2
 1500 d%=d%=c%
 1510 PROClimit(1,a%,25)
 1520 PROClimit(1,b%,25)
 1530 PROClimit(1,c%,80)
 1540 PROClimit(1,d%,80)
 1550 FOR i%=a% TO b%
 1560 MOVE x0%+4+chx%*(c%-1),y0%+1024-20
-chy%*(i%-1)
 1570 PRINT MID$(text$(index%,i%),c%,d%)
;
 1580 NEXT i%
 1590 ENDPROC
 1600 :
 1610 DEF PROClwacliprectangle(b,RETURN
x1%,RETURN y1%,RETURN x2%,RETURN y2%)
 1620 LOCAL x0%,y0%
 1630 PROClwaorigin(b,x0%,y0%)
 1640 x1%=b!24-x0%:y1%=b!28-y0%
 1650 x2%=b!32-x0%:y2%=b!36-y0%
 1660 ENDPROC
 1670 :
 1680 DEF PROCcursor(i%,col%,row%)
 1690 LOCAL x%,y%
 1700 x%=4+chx%*(col%-1)
 1710 y%=1024-48-chy%*(row%-1)
 1720 SYS setcaret,window%(i%),-1,x%,y%,
&1000000 OR 36,-1
 1730 ENDPROC
 1740 :
 1750 DEF PROClimit(lower,RETURN value,u
pper)
 1760 IF value<lower THEN value=lower
 1770 IF value>upper THEN value=upper
 1780 ENDPROC
 1790 :
 1800 DEF FNfinished
 1810 LOCAL finished,i%
 1820 finished=TRUE
 1830 FOR i%=1 TO maxindex
 1840 IF window%(i%)<>-1 THEN finished=F
ALSE
 1850 NEXT i%
 1860 =finished
 1870 :
 1880 DEF FNwhichwindow(handle%)
 1890 LOCAL i%,this%
 1900 i%=1:this%=-1
 1910 FOR i%=1 TO maxindex
 1920 IF window%(i%)=handle% THEN this%=
i%
 1930 NEXT
 1940 =this%
```

Amendments required to make programs run on 32-bit RISC OS 5 (Raspberry Pi)

See the article in the Winter 2024 edition of *Drag 'N Drop*.