DRAG'NDROP

RISC OS **Pi** and all RISC OS 5 machines

January 2016 Volume 7 issue 2 £3.50

The Sound Issue

32bitting sound modules

WAV files

SoundCon32

Module surgery

Python

Mr Frog

The world's only regular RISC OS mag!

Copyright © Drag 'N Drop 2016
Produced on RISC OS computers

This issue has been blessed with contributions from the following people:
Paul Dunnington (Python Primary School)
Jon Robinson (Wimps)
Christopher Dewhurst (everything else)

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

EDITORIAL

Happy new year. This issue is slightly late for January but a trend has emerged where *Drag 'N Drop* comes out just before a show and the next show is the South West show on February 27th. Hope you can come along!

This time we have a wealth of information on getting to know the sound system of your RISC OS Raspberry Pi.

With the article on updating SoundCon app, discussion about WAV files and "32 bitting" sound modules you can jazz up your RISC OS computer with lots of sound effects.

We also have the usual eclectic mix of programming articles including beginner's hexadecimal, Arm code and Python primary school.

Chris.

Christopher Dewhurst

At a glance...

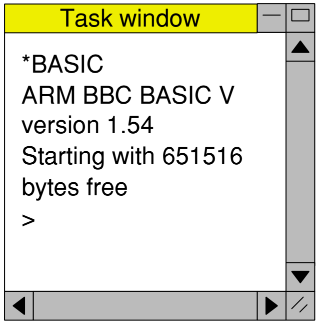
Editorial	1
How do I...?	2
News & Apps	3
Mr Frog	5
Files of the World	7
32 bitting sound modules	13
Module Surgery	15
SoundCon32	20
CDFaker front end	24
Python Primary School	26
Desktop Retro Gaming	32
Wimps	36
&CAFE	41
3D Graphics	43

How do I...?

...get the BBC Basic prompt?

To get the BBC Basic prompt press F12 and type *BASIC and press Return. You can change the screen mode with MODE n where n is a number e.g. MODE 7 or MODE 0. Type AUTO for automatic line numbering. Press Escape to stop and type *SAVE "myprog"* followed by Return to store *myprog* on hard disc.

To return to the desktop type *QUIT. Programs listed in *Drag 'N Drop* are assumed to work on all machines with RISC OS 5 e.g. Raspberry Pi, unless otherwise stated.

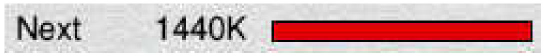


```
*BASIC
ARM BBC BASIC V
version 1.54
Starting with 651516
bytes free
>
```

You can also program and run Basic programs from the desktop. Double-clicking on the filer icon runs it, holding down Shift and double clicking loads it into your text editor.

...open a Task window?

Menu click over the Raspberry icon on the right side of the iconbar and select click on Task window. Or press Ctrl + F12.



Next 1440K

You may need to reserve more memory for the task in which case adjust-click on the Raspberry icon and under Application tasks click and drag the Next slide bar out to the right.

You can also type programs in a task window, hold down Ctrl and press F12. You can't use the cursor editing facility or change MODE, however.

...select the currently selected directory?

Articles may tell you to set the CSD (currently selected directory). Just click menu over filer window and choose Set directory ^W or you can use the !EasyCSD application presented in *Drag N Drop* 6i1.

...open an Applcation Directory?

Application directories begin with a ! called 'pling'. Hold down shift and double click select to open the directory

News and App Updates

South West Show

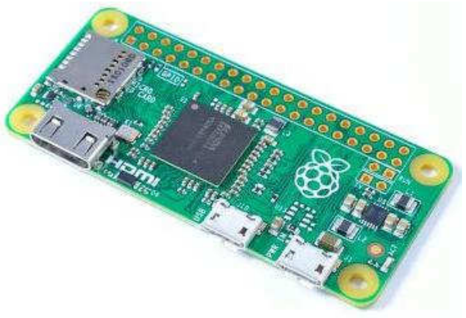
The annual show takes place at the Webbington Hotel near Loxton in Somerset, Saturday 27th Feburary. More details are on the show's website at www.riscos-swshow.co.uk/ If you want the full edition of this issue of *Drag 'N Drop* at a special price of £3 just bring along your memory stick or email address!

New Hardware

Cambridge-based Elesar have developed the basis of a new RISC OS computer. The Titanium motherboard is like an Iyonix on steroids: an ARM Cortex-A15, 2GB Ram, 8 USB sockets, 2 ethernet ports, 2 serial ports and lots more. Prices start at £498 and can be purchased from shop.elesar.co.uk. This is not a complete computer but we've heard rumours that RISC OS developers are thinking about an "out of the box" solution...

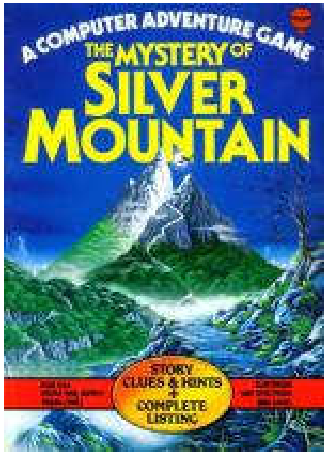
PiZero

Just before Christmas the Raspberry Pi world witnessed the birth of a new baby: the PiZero for under a fiver. They couldn't make 'em fast enough as stocks quickly dried out. Oddly, it's never featured officially on the RISC OS Open website but it will run RISC OS 5, the Release Candidate 14 (RC14) of the RISC OS Rom as featured on the download page www.riscosopen.org/content/downloads/raspberry-pi



Usborne 80s Books

Following in the footsteps of our *55 BBC Micro Books* CD-Rom, publisher Usborne has released 1980s coding books as free PDF downloads at www.usborne.com/catalogue/feature-page/computer-and-coding-books.aspx



If you wish to download the whole PDF in Netsurf we've found it necessary to edit the URL: click Menu over the book cover image, choose Object > Link > Save > Text, drag drop the **open** file onto an Edit window, change the **open?id=** to **uc?export=download&id=** and finally drag drop the modified file onto the URL bar of Netsurf, where you can download and save the PDF file as normal.

For instance the link to *The Mystery of Silver Mountain* changes from https://drive.google.com/open?id=0B2Z4GOoRXHWUeWJnVmlUTGc5NFU to https://drive.google.com/uc?export=download&id=0B2Z4GOoRXHWUeWJnVmlUTGc5NFU

TopModel

Work is progressing to bring back to life the 3D drawing program developed in the 1990s by Paola Castagno, including 32 bitting the program and removing the original copy protection.

The intention is for TopModel to work like a 3D CAD program and support STL file export so that models developed using the program can be 3D printed.

The price of such systems on the PC is still very expensive and it is hoped that the cost of TopModel plus RISC OS hardware to run it on will appeal to 3D developers.

A trial version of the software plus sample models is available for download from Steve Royd-Markers's website at www.markerdesign.be/

Click on *Projecten* > TopModel. Note the Gemini module (used for rendering the models) is still 26 bit so Aemulor is required for everything to work properly. Happy viewing !

32-bit PD apps

Here at *Drag 'N Drop* we are using the services of our resident machine code amphibian and artiste, Mr Frog, to update old PD apps to work on the Raspberry Pi.

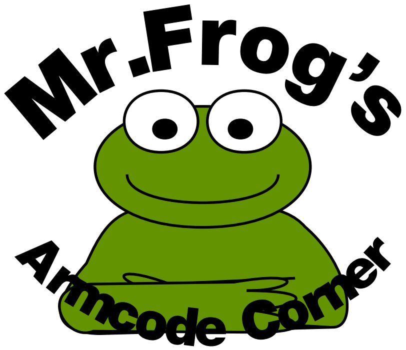
These are appearing on the website at www.dragdrop.co.uk/free. !WaveUtil is a desktop program to convert WAV files to sound modules (WAV files are discussed in this issue's Files of the World) and !DrawCross helps you draw crossword grids. More apps to follow in due course.

StrongEd 4.69

Version 4.69f9 of the advanced text processor has been released fixing a few bugs and now comes with all modes preinstalled. Update your copy here: www.stronged.iconbar.com/

RiscOSM 1.32

The OpenStreetMap (OSM) based vector software is available and so is map data for British Isles, Netherlands and Australasia. Free to registered users. If you haven't bought your copy get along to www.sinenomine.co.uk/software/riscosm/



Now Mr F has finished gloating over the fact his article is the first one in this issue of *Drag 'N Drop* we'll get down to some more serious business.

Last time we met with the Branch-with-Link instruction (BL). Here's a snippet of code which calls a routine *addit* to add the contents of R1 and R2 and store the result in R0:

```
BL addit ; call addit routine
; return to here
...
.addit
ADD R0,R1,R2 ;R0=R1+R2
MOV PC,R14 ;R15=R14
```

When the computer is at the *BL addit* instruction it stores the return address in R14 just before skipping to *addit* and that's why the there's that MOV PC,R14 at the end to put R14 back in the Program Counter so it knows where to go back to.

BL is like the JSR instruction in steam-powered assemblers such as the 6502, and MOV PC,R14 is like RTS (Return from Subroutine)

But that wouldn't work if we then want to branch to another subroutine within *addit.* The return address to *addit* would be stored in R14 and the original return address overwritten.

One way round it would be for the subroutine to make a copy of R14. Take a look at Program 1.

```
   10REM Mr.Frog's Prog1
   20REM Drag 'N Drop Jan 2016
   30DIM code 200
   40FOR pass=0 TO 3 STEP 3
   50P%=code
   60[OPT pass
   70MOV R10,R14
   80BL sub1
   90MOV PC,R10
  100.sub1
  110MOV R11,R14
  120SWI "OS_WriteS"
  130EQUS "Mr Frog "+CHR$0
  140ALIGN
  150BL sub2
  160MOV PC,R11
  170.sub2
  180MOV R12,R14
  190SWI "OS_WriteS"
  200EQUS "in a Yellow Sub"+CHR$0
  210ALIGN
  220MOV PC,R12
  230]
  240NEXT
  250CALLcode
```

The link register R14 is copied to R10 before *sub1* is called. The first thing that *sub1* does is to preserve its own copy of R14 in R11 then call SWI OS_WriteS, an operating system routine to print the string immediately following. The string has to be terminated by a zero byte (CHR$0) and the ALIGN tells the assembler to jump to the next word (four-byte) boundary because addresses can only end in 0, 4, 8, or &C.

sub1 calls *sub2* and then restores its copy of R14 from where we stored it in R11. *sub2* is exactly the same as *sub1* except R12 is used to hold a copy of R14.

Now this is all very well but we've tied up R10, R11 and R12 which means they're not available to other parts of the machine code program (if we were writing a big one).

The Frog family uses a well-known trick in the Armcode which involves putting registers onto a stack.

```
   10REM Mr.Frog's Prog2
   20REM Drag 'N Drop Jan 2016
   30DIM code 200
```

Files of the World

4 WAV files

In this instalment we'll take a look at the world of sound wave files, usually known as sound samples.

A sound sample is a short (usually just a few seconds long) recording of a sound. The bytes in the file represent what the sound wave is doing over that brief time.

A WAVe file is a common type of sound sample and its filetype is &FB1 (WaveForm) on RISC OS. The file icon looks like one of the following:

I'll recap on how sound 'works' then look at the innards of the WAV file format and finally present a couple of BBC Basic demonstration programs which display information on WAV files and plot sound samples on the screen.

Sound waves are movements of air particles, compression and decompression of the surrounding air, caused by movement of something physical like beating of a drum or plucking of a string.

These air particles travel in waves until they reach a diaphragm, a tiny sheet of very thin material in a listening device.

The listening device is usually our ears or it could be a microphone. The slight movement of the sheet throughout the sound is recorded in numbers and stored as bytes in the sound sample file.

To reproduce the sound, the same numbers are used to vary the position of a cone inside a speaker, thereby recreating the original sound wave.

Now, the position of the diaphragm or speaker cone changes literally thousands per second. You will have heard audio enthusiasts go on about *sample rates*. This is the number of times per second the speaker cone moves to pump sound waves into the air.

44,100 samples a second, sometimes abbreviated to 44.1k or 44.1 kHz (kilohertz, cycles per second) is a typical sample rate.

Figure 1 shows different "graphs" of soundwaves (plotted using Program 2 the listing for which is presented at the end).

The graph for the raindrops on roof jumps about with sharp spikes. The piano graph dies off at the end because after you strike a note on the piano the sound fades away. The flute is quite smooth and thick. The crackling fire is jagged but confined vertically - it has a small *amplitude* compared to the rain.

32 Bitting Sound Modules

A sad fact of modern day RISC OS machines like the Raspberry Pi is that old sound modules - 26 bit sound modules, that is - don't work.

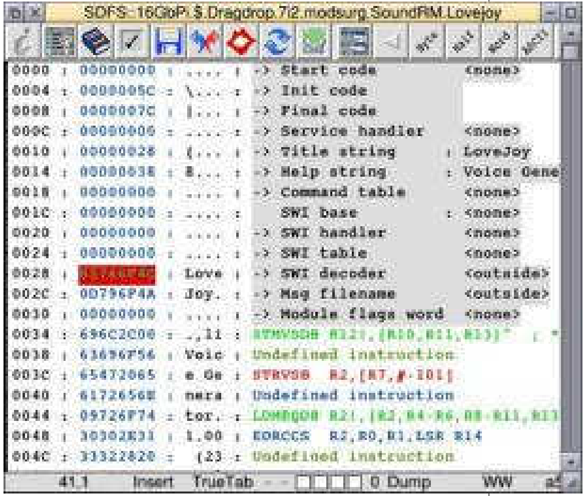
If you double click to install, say, a module called Applause you will get the dreaded "Application may have gone wrong" error and clicking on the Describe button gives "Module 'Applause' is not 32-bit compatible"

Sound modules are usually generated by an application such as !SoundCon rather than assembled from source. So generally there no source code to modify and reassemble it to make it 32 bit compliant.

So we have poke around in the binary code itself. Specifically, we have to add some extra words to the module header and update four pointers: the pointer to start code, finalisation code, title string and help string.

Sound modules, by and large, don't provide extra SWIs or extra star commands so we don't have to worry about changing anything more than those four pointers. StrongEd provides a good disassembler which we'll use to achieve this.

First of all load in StrongEd. Drag a 26-bit module onto the iconbar icon. You should get a display like figure 1. If not, check you have the latest version of StrongEd available from stronged.iconbar.com.



```
0000 : 00000000 : .... : -> Start code              <none>
0004 : 0000005C : \... : -> Init code
0008 : 0000007C : |... : -> Final code
000C : 00000000 : .... : -> Service handler         <none>
0010 : 00000028 : (... : -> Title string         : LoveJoy
0014 : 00000038 : 8... : -> Help string          : Voice Gene
0018 : 00000000 : .... : -> Command table           <none>
001C : 00000000 : .... :    SWI base             : <none>
0020 : 00000000 : .... : -> SWI handler             <none>
0024 : 00000000 : .... : -> SWI table               <none>
0028 : [illegible] : Love : -> SWI decoder          <outside>
002C : 0D796F4A : Joy. : -> Msg filename            <outside>
0030 : 00000000 : .... : -> Module flags word       <none>
0034 : 696C2C00 : .,li : STMVSDB R12!,{R10,R11,R13}^
0038 : 63696F56 : Voic : Undefined instruction
003C : 65472065 : e Ge : STRVSB  R2,[R7,#-101]
0040 : 6172656E : nera : Undefined instruction
0044 : 09726F74 : tor. : LDMNEDB R2!,{R2,R4-R6,R8-R11,R13}
0048 : 30302E31 : 1.00 : EORCCS  R2,R0,R1,LSR R14
004C : 33322820 :  (23 : Undefined instruction
```

Figure 1

The area shaded grey is the module header. Click on the second column of blue hexadecimal numbers next to offset 0028, a block of red will start to flash as in figure 1.

Press the zero key and down arrow. Repeat (zero key, down arrow) three times.

This inserts four blank words into the module. You'll see the grey area expand. 32-bit modules must have a module flag pointer at offset &30.

We can put the 32-bit module flag at offset &34, it has to have bit one set, i.e. the word has value 1. Click on "Insert" at the bottom of StrongEd's window to change to "Overwr". Position the flashing red cursor block on the word at offset &30 and type 34 (that is the 3 key and the 4 key).

Press the down cursor once (so the red block is on the word at offset &34) and type 1. Your display should look like figure 2.

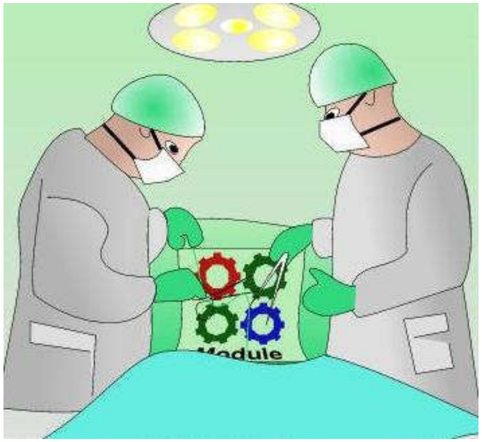
Module Surgery

When 32-bit RISC OS 5 arrived in the early 2000s the old 26-bit relocatable modules wouldn't work on the new machines. You get the dreaded *Module 'X' is not 32-bit compatible* error message if you try.

Modules are written in machine code. The machine code instruction set in RISC OS 5 computers differs from that in older RISC OS computers. So that the operating system doesn't try to run 'illegal' machine code and result in your computer crashing, the layout of the module *header* (a short section of code at the beginning of the module) has been changed on RISC OS 5.

The system checks the header and if it doesn't conform to the new standards the module will not be loaded.

The module header on RISC OS 5 is longer and all of the words in the header (a word is four bytes of memory) are compulsory. This is in contrast to old-style modules the header was



shorter with several optional words.

Figure 1 compares the old and the new format. As you can see, 26 bit modules could consist of seven words (11 if it provided its own SWI calls) but the header of a 32 bit module must *always* be 13 words long. All you need to know about SWIs for the time being is that if a 32-bit module doesn't provide any new SWI calls then the four SWI words are zero.

It may be of course that aside from the header the module is made up of completely legal and

Off set	RISC OS 5 (32 bit) Offset to...	Older RISC OS (26 bit): Offset to...
&00	Start-up code	
&04	Initialisation code	
&08	Finalisation code	
&0C	Service call code	
&10	Title string	
&14	Module *HELP string	
&18	Help and command decoding table	
&1C	SWI base number for this module	SWI base number for this module (optional)
&20	SWI handling code (compulsory)	SWI handling code (optional)
&24	SWI decoding table (compulsory)	SWI decoding table (optional)
&28	SWI decoding code compulsory)	SWI decoding code (optional)
&2C	Messages filename	
&30	32-bit flag	

Figure 1: 26-bit and 32-bit module headers

compatible code which works in the new world. All you then have to do is tweak the module header.

If you have the 'source' listing (the assembly language program) for the module then all that is required is to insert a few extra words into the module header, assemble it and save the new

SoundCon32

SoundCon is a freeware sound sample converter dating from the 1990s and it allows conversion between over a dozen popular formats including RISC OS Sound Modules. In this article I'll show how to update the application to run on the Raspberry Pi.

I'll present short user's guide to SoundCon first then proceed to a more technical discussion for those interested in what's involved 'behind-the-scenes' to make the the application 32 bit compatible, i.e. work with RISC OS 5.

SoundCon32 User Guide

Double click on !PlayIt. Please refer to the technical discussion on where to obtain !PlayIt and what to tweak in the the Run file to load the correct driver.

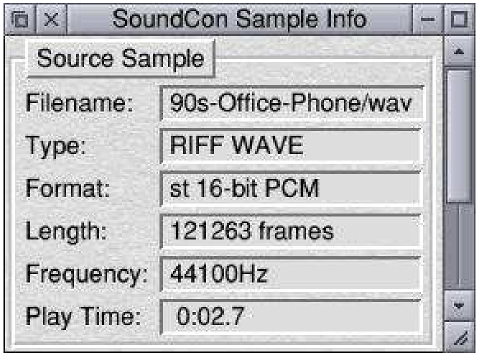
This does not load anything on the iconbar but installs the necessary 'drivers' to play back sound samples.

Double click on the !SoundCon app itself to put it on the iconbar.

Next find a sound sample, there are hundreds of WAVe files on the internet. Try freewavesamples.com.

Netsurf should set the filetype to &FB1 from the /wav extension but if it hasn't you will need to manually by clicking Menu over the file and choosing File 'xxx' > Set Type > type '&FB1' in the dialogue box.

A window will pop up giving information on the sound sample

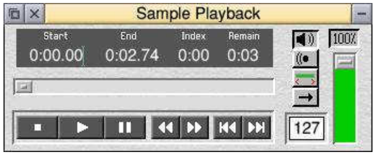

SoundCon Sample Info

Source Sample

Filename:	90s-Office-Phone/wav
Type:	RIFF WAVE
Format:	st 16-bit PCM
Length:	121263 frames
Frequency:	44100Hz
Play Time:	0:02.7

This window may also be called up by clicking Menu on the iconbar icon and choosing *Sample info*.

Click Menu over the *Sample info* window window you will get three options: Convert, Play Sample and Clear sample. Move right across the Play Sample option to bring up the Sample Playback window:



This window may also be called up by choosing Play... from the icon bar menu.

To play the sample click once on the rewind icons ⏪ or ⏮ and then click the play ▶ icon. (I am not sure why it's necessary to click the rewind icon but if you try clicking play immediately the sound doesn't play.)

The controls on the right hand side of the window are self explanatory.

To convert the sound sample choose Convert... from the

A Front End for CDFaker

CDFaker is a relocatable module which allows you to access CD-Roms stored as image files rather than physical CDs.

Typically you click on the CD icon (on the left hand side of the iconbar) whic brings up a filer window on the CD image just like a real CD Rom. You can download CDFaker for free from www.huber.net.de/cdfaker.html.

Coming as a bare module it isn't very user-friendly and actually mounting a CD-Rom image involves some scary things like Obey files.

In this article I'll show you how to create a "front end". All you need then do is double click a CD image then single click on your CD drive icon and away you go. If you want to 'change' CDs just double click on another CD image.

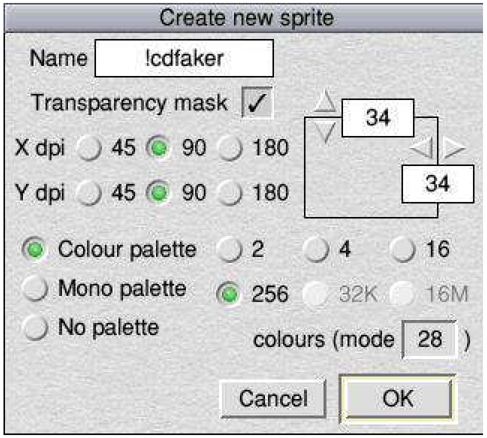
First of all create a blank application directory. Do this by calling up a filer window on your hard disc (it could be in the root directory - click on the :0 icon on the iconbar). Click menu over the filer display and choose New directory > type !CDFaker in the box and press Return. Hold down shift and double click !CDFaker to open the directory. Download the CDFaker module from the website and put it inside !CDFaker.

We're going to design a couple of CD sprites using Paint so load up Paint by clicking on Apps and double clicking on !Paint.

Designing a picture of a CD isn't hard as it's just a circle with a few coloured segments in it. Click on Paint's iconbar icon and in the Create new sprite window, in the Name box type !**cdfaker** (all lower case), 256 colours, Colour palette and the size should be 34 x 34 pixels.

If you're stuck there are hundreds of designs on the internet you can use, just search for "cd icon".

For my design I've chosen a light grey colour (number 252),


Create new sprite

Name: !cdfaker

Transparency mask [✓]

X dpi: 45 [✓ 90] 180

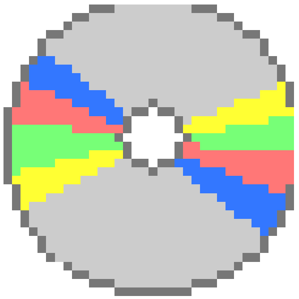
Y dpi: 45 [✓ 90] 180

Width: 34, Height: 34

[✓ Colour palette] Mono palette No palette

2 4 16 [✓ 256] 32K 16M

colours (mode 28)



used the filled circle tool to draw the disc, the filled arc tool put in some coloured segments in pastel colours, unfilled circle tool to put a dark grey border around the disc and filled in the gap between the circle and surrounding square with the mask

Python Primary School

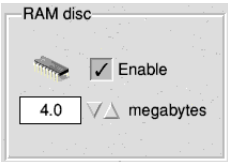
You will know that to set the file type of all files in a directory you click menu in the directory viewer > select all then click menu again and choose Selection > Set type then type in either number or name.

The problem is that any sub directories remain untouched. So this term we will work on a program to set the file type of *all* files with the extension /py to &AE5, Python. We'll also learn about global variables.

If you don't already have one, set up a RAM disc by clicking Select on the SD card icon on the icon bar then double click Select on !Boot to bring up the Configuration viewer. Click the Discs > tick the Enable box for the RAM disc.

You can alter the size by the up and down icons or click select in the writable icon to delete and type in a value. 4 megabytes will be ample for our purposes.

Click Set and you now have a RAM icon on the icon bar. It will



RAM disc
✓ Enable
4.0 megabytes

be there every time your Pi starts up unless you untick the Enable icon and click Set in !Boot > Configuration > Discs.

Our first program this term is **RiscOSext1/py** and is listed below. The line numbers are for reference only and should not be typed in.

If you are using StrongEd you can display line numbers by clicking menu on the document window > Basemode > Choices > LineNos > and clicking **Physical** radio icon under Type. (In Edit there is no such facility but pressing F5 will bring up a window telling you the line you are on.)

RiscOSext1/py

```
 1# RiscOSext1/py
 2# List the sub directories in th
e
 3# given directory.
 4#
 5# by P. Dunnington
 6# Drag N Drop Jan 2016
 7
 8import sys, os
 9
10def riscos_ext(d='RAM::RamDisc0.
$'):
11    if os.path.isdir(d):
12      print 'Directory exists'
13      print os.listdir(d)
14    else:
15        print "Can't find Direct
ory", d
16
17riscos_ext()
```

Lines 1-6 are comments giving information on what the program does, the author and date. Line 8 imports the modules we will be using. We have met **def** in line 10

Desktop Retro Gaming

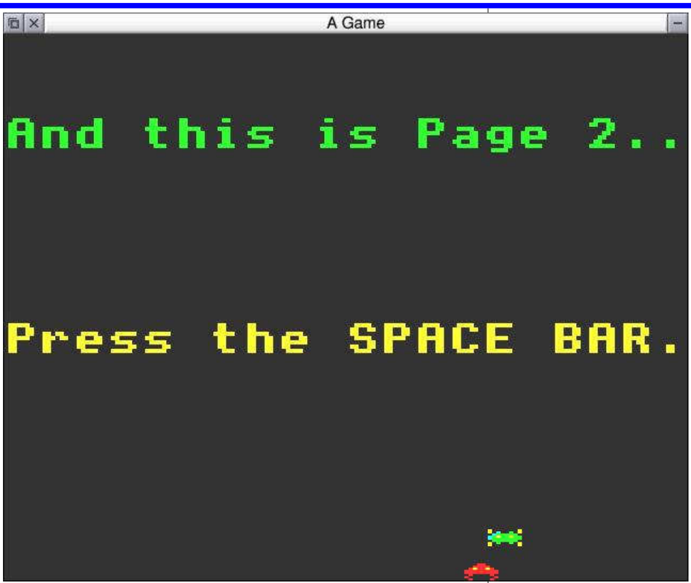
In the Autumn 2015 issue of *Drag 'N Drop* I demonstrated some techniques you can use to run BBC Basic games, originally written to run as single tasking, in a window on the desktop.

An important point is that the Wimp Manager can request your window to be redrawn at any time. There has to be a way of knowing if the game is in the middle of moving aliens or on the title screen or perhaps displaying high scores.

Type in Program 1. PROC redraw, PROCgetorigin, PROC window and PROCerror are exactly the same as last time.

The way the sprites are set up in lines 1110-1250 is different. Instead of generating a sprite file separately and loading it in, the sprites are stored directly in memory with a couple of SpriteOp SYStem calls.

SpriteOp &109 initialises a user sprite area at *area%* as before. *SYS "OS_SpriteOp", &10F,area%, name$,0,x,y,mode* creates a blank sprite called *name$* of (*x*,*y*) dimensions in mode *mode* in the sprite *area%*.



```
And this is Page 2..

Press the SPACE BAR.
```

SYS "OS_SpriteOp",&12A, area%,name$,x,y,colour,0 fills in a pixel of *colour* at position (*x*,*y*) in the sprite (y=0 being the bottom) whose name is *name$*.

The *colour* is 0 to 63 for 256 colour modes such as mode 13. Each Ascii character in the sprite data in lines 1410-1570 is the colour number with 63 added so ?=0, @=1, A=2 and so on. Table 1 lists the eight BBC-style colours and the corresponding Ascii code for data stored in this format.

The loop in lines 1200-1260 simply extracts the letters using MID$, subtracts 63 from the ASCii value and calls Sprite_Op &12A to put this pixel value into the sprite.

BBC COLOUR / GCOL	Mode 10/15 COLOUR / GCOL	ASCII char	Colour number (!Paint)
0	0	?	0
1	3	B	20
2	12	K	96
3	15	N	116
4	32	_	128
5	51	r	156
6	60	{	232
7	63	~	252

Table 1

RISC OS Programming

Part 8 - Determining the File Path (again)

In the last instalment of this tutorial, we learnt how to read the file name icon in the Save window, use Wimp_Send Message to obtain the file path to the target directory and update the writeable icon in the Save window.

But we were left with the problem. If we dragged the file icon to the destination window *twice* it messed up the file path.

In this instalment we are going to deal with this problem and learn a bit more about string handling in RISC OS.

The updated program is listed at the end of the article. Save it inside *!PDF.Stage8* and update the !Run file so the last line reads:

```
Run <Obey$Dir>.Stage8
```

Stripping off the File Path

To stop the program messing up the file path we have to go back to PROC FileIconDropped and strip off any path component that has been added before the file name *before* it is sent as a message to the operating system.

After we have done Wimp_Get IconState and transferred what's in the text field to IconContents$ we need to call a new function JustFileName which will strip any path component from the front of the text string, leaving just the file name at the end.

```
  620DEF PROCFileIconDropped
  ...
  700IF DestinationHandle%<>-2 AND
 DestinationHandle%<>SaveHandle% T
HEN
  ...
  730 SYS "Wimp_GetIconState",,Blo
ck2%
  740 IconContents$=$(Block2%!28)
  770 REM> Remove path from file n
ame, if there is one.
  780 IconContents$=FNJustFileName
(IconContents$)
  ...
  900ENDPROC
```

It would also be a good idea to set up a file name, if none has been entered:

```
  810 IF IconContents$="" THEN Ico
nContents$="TextFile"
```

We can now define the function JustFileName:

```
  940DEF FNJustFileName(IconString
$)
  960Pointer%=INSTR(IconString$,".
")
 1030WHILE Pointer%>0
 1040 IconString$=MID$(IconString$
, Pointer%+1)
 1060 Pointer%=INSTR(IconString$,"
.")
 1080ENDWHILE
 1100=IconString$
```

This function is called every time we drop a file icon onto a destination window, after the contents of the text field have been read into *IconContents$*.

Once that text string has been passed to JustFileName, line 1060 checks to see if there any point (full stop) characters in it.

If there are none it is just a file name. *Pointer%* will be set to zero and the function will just return the string back to FileIcon Dropped without making any changes.

If there are any point

&CAFE

Welcome back to the Hexadecimal Cafe. Take a 5EA7 (seat) and have a C0FFEE (coffee) while we get to grips with words, bytes and go from low byte to high byte...

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

We learned that one hexadecimal digit represents 16 possible values, which is the same as saying we can store the decimal numbers zero to 15 in one hexadecimal digit.

A hexadecimal digit (or hex digit for short) is a group of four binary numbers or four *bits*. I've repeated the table above to refresh your memory.

When we want to count above 16 we string together another hex digit:

Decimal	Binary	Hex
16	0001 0000	10
17	0001 0001	11
18	0001 0010	12
19	0001 0011	13

Continuing in this fashion we can count all the way up to FF. Which is 255 in decimal. It also happens to be the maximum value you can store in a *byte.* A byte holds 256 possible values from zero to 255 in decimal or zero to &FF in hex.

When we communicate with the computer, if we want to tell it we mean hexadecimal then we prefix the hex with &.

Now, if we carry on counting above 255 something happens. &100 or 256 is the next number after 255 and it is three hex digits. &101 (decimal 257) also is three digits and so on. That means we need another *byte* to store the leftmost digit. Look at this the number on the right.

&1|01

If you draw an imaginary line between the &1 and 01 you can see that one byte holds the 1 on the left and another byte the 01 on the right. Okay so how about this then:

&C0|FF|EE

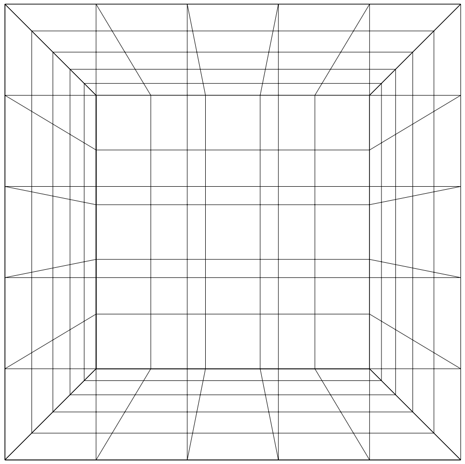
We've got six hex digits. The

3D Wireframe Graphics

So far we've seen how to set up basic shapes for rectangular grids and concentric circles. With various transformations you were able to combine a series of grids into perspective views of a cube.

Unfortunately the gremlins crept in and some procedure definitions were missing from the last issue. First of all we present the full listing (with missing procedures) for the cube program:

```
  100MODE 15
  110PROCinit
  120len =50 : p = 100 : n = 5
  130REPEAT
  140v = FNgetposition
  150IF v THEN PROCtorus(50,18,100
,18)
  160UNTIL NOT v
  170CLS
  180END
  500DEF FNgetposition
  510INPUT "Eye position (x,y,z)",
x,y,z
  520PROCposition(x,y,z)
  530CLS
  540= (x <> 0) OR (y <> 0) OR (z
<> 0)
 2000 DEF PROCtorus(r,n,r2,n2)
 2010 LOCAL ka,kb,kc,nc
 2020 PROCorigin(0,0,0)
 2030 PROCyvector(0,0,1)
 2040 ka =2*PI/n2 : nc = n2
 2045 IF r2=0 THEN nc = nc DIV 2
 2050 kb = 0
 2060 FOR kc = 1 TO nc
 2070 PROCxvector(COSkb,SINkb,0)
 2080 PROCcircle(r2,0,r,n)
 2090 kb+= ka
 2100 NEXT
 2110 PROCxvector(1,0,0)
 2120 PROCyvector(0,1,0)
 2130 ka = 2*PI/n
 2135 IF r2 = 0 THEN ka = ka/2
 2140 kb = 0
 2150 FOR kc = 1 TO n
 2160 PROCorigin(0,0,r*COSkb)
 2170 PROCcircle(0,0,r2 + r*SINkb,
n2)
 2180 kb+= ka
 2190 NEXT
 2200 ENDPROC
 5000DEF PROCgrid(xa,ya,lw,lh,nx,n
y)
 5010ja = lw/nx
 5020xs = xa
 5030FOR jb = 0 TO nx
 5070PROCline(xs,ya,xs,ya + lh)
 5080xs+= ja
 5090NEXT
 5100ja=lh/ny
 5105 ys = ya + lh
 5110FOR jb = 0 TO ny
 5150PROCline(xa + lw,ys,xa,ys)
 5160ys-= ja
 5170NEXT
 5180ENDPROC
 5190:
 6000DEF PROCcircle(xs,ys,r,n)
 6010IF n = 0 THEN n = 20+ INT(r/1
0)
 6020ja = 2*PI/n
 6050PROCmove(xs + r,ys)
 6060jb = 0
 6070FOR jc = 2 TO n
 6080jb+= ja
 6110PROCdraw(xs + r*COSjb, ys + r
*SINjb)
 6120NEXT
 6150PROCdraw(xs + r,ys)
 6160ENDPROC
 8000DEF PROCposition(x,y,z)
 8010LOCAL wv,wu,g,sg,cg,xt,yt
 8020xv = x
 8030yv = y
 8040zv = z
 8050wv = yv*yv + zv*zv
 8060pv = SQR(xv*xv + wv)
 8070wv = SQR wv
 8080IF pv = 0 THEN ENDPROC
 8090xu = xv/pv
 8100yu = yv/pv
 8110zu = zv/pv
 8120wu = wv/pv
 8130REM eye orientation
 8140g = FNatan(xv*yv,zv) + FNatan
(yv,xv)
 8150sg = SIN g
 8160cg = COS g
 8170REM rotation matrix
 8180r11 = wu*cg
 8190r21 = -wu*sg
 8200r31 = -xu
 8210r32 = -yu
 8220r33 = -zu
 8230r34 = xv*xu + yv*yu + zv*zu
 8240IF wu=0THEN 8340
 8250xt = xv*wu - (yv*yu + zv*zu)*
xu/wu
 8260yt = (yv*zu - zv*yu)/wu
 8270r12 = (zu*sg - xu*yu*cg)/wu
 8280r13 = (-yu*sg - xu*zu*cg)/wu
 8290r14 = cg*xt + sg*yt
 8300r22 = (zu*cg + xu*yu*sg)/wu
 8310r23 = (-yu*cg + xu*zu*sg)/wu
 8320r24 = -sg*xt + cg*yt
 8330ENDPROC
 8340:
 8340REM special case on x-axis
 8350r12 = -1
 8360r13 = 0
 8370r14 = 0
 8380r22 = 0
 8390r23 = 1
 8400r24 = 0
 8410ENDPROC
 8450DEF FNatan(a,b)
 8460IF b <> 0THEN =ATN(a/b) ELSE
=PI/2
 8500DEF FNtrans(x,y)
 8510LOCAL x1,y1,z1,x2,y2,z2
 8520REM 2-D to 3-D
 8530x1 = t11*x + t12*y + t13 + xo
 8540y1 = t21*x + t22*y + t23 + yo
 8550z1 = t31*x + t32*y + t33 + zo
 8560REM object to eye
 8570x2 = r11*x1 + r12*y1 + r13*z1
 + r14
 8580y2 = r21*x1 + r22*y1 + r23*z1
 + r24
 8590z2 = r31*x1 + r32*y1 + r33*z1
 + r34
 8600IF z2 < zmin THEN =FALSE
 8610REM 3-D to 2-D
 8620x3 = d*x2/z2
 8630y3 = d*y2/z2
 8640=TRUE
 8650:
 8800 DEF PROCoffset(x,y,z)
 8810 xo = x
 8820 yo = y
 8830 zo = z
 8840 ENDPROC
 9000DEF PROCinit
 9010CLS:CLG
 9020xmax = 1280: ymax = 1024
 9030xmid = xmax/2 : ymid = ymax/2
 9035ORIGIN xmid,ymid
 9040zmin = 1
 9042INPUT"Enter projection plane
distance",d
 9045IF d <=0THEN d=1000*zmin
 9050PROCxvector(1,0,0)
 9060PROCyvector(0,1,0)
 9070PROCorigin(0,0,0)
 9080 PROCoffset(0,0,0)
 9085CLS
 9090ENDPROC
 9100DEF PROCmove(x,y)
 9120IF FNtrans(x,y) THEN MOVE x3,
y3
 9130ENDPROC
 9200DEF PROCdraw(x,y)
 9220IF FNtrans(x,y) THEN DRAW x3,
y3
 9230ENDPROC
 9240:
 9500DEF PROCline(xs,ys,xe,ye)
 9510PROCmove(xs,ys)
 9540PROCdraw(xe,ye)
 9550ENDPROC
 9600DEF PROCxvector(dx,dy,dz)
 9610t11 = dx
 9620t21 = dy
 9630t31 = dz
 9640ENDPROC
 9700DEF PROCyvector(dx,dy,dz)
 9710t12 = dx
 9720t22 = dy
 9730t32 = dz
 9740ENDPROC
 9800DEF PROCorigin(x,y,z)
 9810t13 = x
 9820t23 = y
 9830t33 = z
 9840ENDPROC
```



We now make minor changes to the program developed so far to extend the range of wireframe images you can draw.

It's very easy to set up multiple images. As you put the program through its paces you'll have noticed some spectacular perspective views when the viewing distance **d** is small - 100 to 400 say - whereas there is little perspective effect when **d** is of the order of thousands. You can continue this exercise so it looks