Displaying documentation

Introduction

I have seen various friendly ways of displaying a PDF on the internet, particularly where it contains a catalogue and you only need to find the relevant page. A pair of left- and right-hand pages are usually shown with turned up corners to allow you to move forward and back a few pages. Also an index with hot links to the relevant page.

Some have neat graphics showing a page being turned. So what I wanted was a file full of page images, which RISC OS could render, derived from a PDF file with added links.

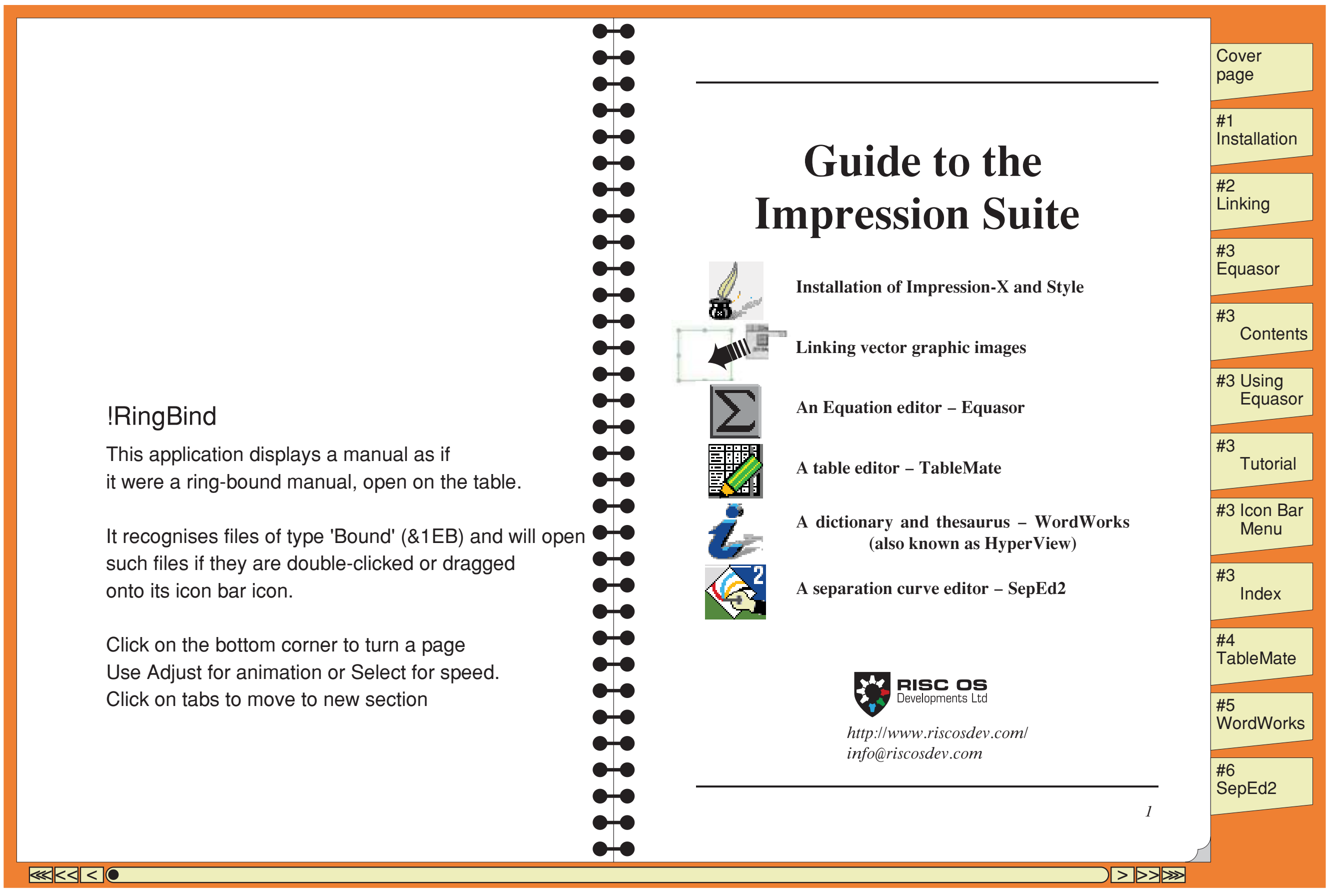
First stage

The first stage was to extract the contents of a PDF file in a form that could be held in memory as individual pages and rendered, on demand, by RISC OS: the obvious format was as Draw files.

At this stage the simplest approach would be a large Draw file that would render as an image of the background (the open covers of a ring-bound file) and could be held in memory, with individual pages held within it, each as a Draw file. Pointers in the Draw file would indicate where the Draw file for each page began and ended.

Can this be done in a 'Draw' file?

Yes. You can hide data inside a 'tag' object within a Draw file. The hidden data within a 'tag' object comprises a single 'ordinary' draw object (which is rendered), a 'tag' number and an arbitrary number of bytes of data, padded to a word boundary. Data within a tag object of a particular tag number are recognised by the application that owns that tag number.



!RingBind

This application displays a manual as if
it were a ring-bound manual, open on the table.

It recognises files of type 'Bound' (&1EB) and will open
such files if they are double-clicked or dragged
onto its icon bar icon.

Click on the bottom corner to turn a page
Use Adjust for animation or Select for speed.
Click on tabs to move to new section

Guide to the
Impression Suite

Installation of Impression-X and Style

Linking vector graphic images

An Equation editor – Equasor

A table editor – TableMate

A dictionary and thesaurus – WordWorks
(also known as HyperView)

A separation curve editor – SepEd2

RISC OS
Developments Ltd

http://www.riscosdev.com/
info@riscosdev.com

1

The background image showing turned corner, index tabs and ring binding, with the cover page added.

4 bytes = object type
4 bytes = object length
16 bytes = bounding box of object
object-specific data

The format of a draw object.

Note that all object types except 0 (font table) have space reserved for a bounding box.

If you include a 'tag' draw file object (object type 7) after the font table object at the start of the Draw file, it has the same format as other draw objects and its object-specific data comprise a tag identifier word (numbers allocated by ROOL), followed by a single draw object and then by a tag-dependent amount of data, the lengths of the draw object and the tag object thus allow extra space to be 'hidden' between them.

When a 'tag' object is rendered, then the object within it is rendered. As that can be a group object, itself containing an arbitrary number of objects, you can put the whole Draw file, apart from the font table, inside a tag object.

In the columns below is an example where the background image is a large Draw file containing all the individual memory blocks:

①	40 byte header - gives main parameters
②	font table object, length aaaa = bbbb - 0028
③	tag &1401 object, length cccc = ffff - bbbb
④	group object, length 0024 = ff40 - ff1C
⑤	Draw file for page 1, length = ptr2 - ptr1
⑥	Draw file for page 2, length = ptr3 - ptr2
⑦	Draw file for last page, length = pppp - kkkk
⑧	tag &1402 object, length gggg = pppp - ffff
⑨	Valid Draw file, 0000 to ffff
⑩	Valid Draw file, 0000 to pppp

Note:
bb04 = bbbb + &0004
cccc = ffff - bbbb = (nn+1)*&0010
eeee = bb1C + dddd
mm = margin in draw units
nn = maximum page number
ww = maximum page width in draw units
hh = maximum page height in draw units
ptrn = start of Draw file for page n
ptr1 = ff40 = bbbb + cccc+ &40
z - bits 30, 29, 28 mark odd pages where chapter, section or sub-section start

	offset	→	data
	0000	→	Draw file header
	000C	→	'Bind'
①	0010	→	ffff offset to end of first tag
	0014	→	qqqq max page size
	0018	→	Draw file bounding box
	0028	→	0000 (font table object)
②	002C	→	aaaa length of font table
	bbbb	→	0007 (tag object)
	bb04	→	cccc (length of tag)
	bb08	→	bounding box
	bb18	→	tag # &1401
	bb1C	→	0006 (group object)
⑨ ③	bb20	→	dddd (length of group object)
	bb24	→	bounding box & name of group
	bb40	→	Draw objects to render backgnd
	....		
	eeee	→	hidden bytes inside tag
	ee00	→	0000,00mm,00ww,00hh
	ee10	→	0001,z0nn,ptr1,ptr2
	ee20	→	0002,z0nn,ptr2,ptr3
	....		
	een0	→	00nn,z0nn,ptrn = kkkk,pppp
⑩	ffff	→	0007 (tag object)
	ff04	→	gggg (length of tag)
	ff08	→	bounding box (all zero)
	ff18	→	tag # &1402
	ff1C	→	0006 (group object)
⑧ ④	ff20	→	0024 (length of empty group)
	ff24	→	bounding box (all zero)
	ff34	→	name of group object (12 char)
⑤	ff40	→	Draw file for page 1
⑥	jjjj	→	Draw file for page 2
	....		
	kkkk	→	Draw file for last page
⑦	pppp	→	End of 'Bound' file

Above left: *'Bound' file is either loaded as a whole (0000-pppp) or part (0000-ffff) plus relevant pages*

Above right: *Content of a 'generalised Bound' file (type &1EB) which can be rendered as a Draw file* pages as separate Draw files with pointers to the offset inside the file where each page starts and ends.

If the whole 'Bound' file can be accommodated in the available Wimpslot, it is clearly faster to do this and it will appear, as a whole, as a valid Draw file. In this case the pointers to each page simply point directly to memory as an offset from where the 'Bound' file was loaded. The 'Bound' file can, however, be quite large.

In most cases it will be more efficient to keep the 'Bound' file open and load just the first part (from 0000 to ffff, which renders as the background and will also be seen, as a whole, as a valid Draw file) and then load just four pages at a time into memory using the pointers as offsets into the 'Bound' file, from which a section of the file can be selected for loading into memory, as and when required.

Two pairs of pages will normally be loaded: the left hand page and the following page as well as the right hand page and the preceding page. This provides the data for turning pages forward and backward.

An example should make this clear: let us consider a move from pages 2 and 3 to pages 14 and 15. From now onwards I'll refer to a pair of pages by the odd page number, the even page being that preceding it. First the pointers will be set up so that the left hand pair of pages is page 3 and the right hand pair is page 15.

Pages 2 and 3 are on view and as the move takes place, page 3 lifts and narrows, revealing page 15 below it. As it turns over, page 15 will be fully uncovered and page 14 will start to obscure page 2. On completion the right hand pair of pages, page 15, will be the only pages visible.

On a backward move, it will be the left hand pair that remain visible.

Provided the pointers to the page data and the page data themselves are changed in a single operation, then they will be valid for the next redraw request. This implies that a call to SYS "OS_GBPB" (to load page data) will not be interrupted by a redraw request.

Turning forward shows (for example) pages 2 and 3 with page 3 lifting and narrowing to reveal page 5 beneath and then page 4 widening to obscure page 3 until just pages 4 and 5 are shown.

Making a start

Let's not worry about using lots of RAM at this point. I wanted to have a 'background' image that looked like the open covers of a ring-bound file onto which a left and right hand page would be rendered. A single Draw file including a background image with space inside it where the individual Draw files for each internal page could be hidden looked like a good start.

Using SYS "DrawFile_Render" with pointers to the start and end of the whole 'Bound' file (0000 to pppp) held in memory would render the background and with pointers set to the start and end of a particular page, then just that page would be rendered.

Version 0.06 of !RingBind does just this. On Virtual Risc PC (or on 32 bit platforms with Aemulor running) it cannot therefore show the larger manuals. Version 0.10 of !RingBind will use less memory if the 'Bound' file permits this and should therefore run on VRPC.

Running the application !RingBind (by double-clicking its icon in 'Apps' for example) will load it with a Wimpslot between the minimum and maximum setting in its '!Run' file according to the current value of the 'next' slot.

Double-clicking a 'Bound' file uses the normal RISC OS protocol to offer a message to any application that can load

```
       offset   → data
       0000     → Draw file header
       000C     → 'Bind'
  ①    0010     → 7364 offset to end of first tag
       0014     → 134FF4 max page size
       0018     → Draw file bounding box
       0028     → 0000 (font table object)
  ②    002C     → 001C length of font table
       0044     → 0007 (tag object)
       0048     → 7320 (length of tag)
       004C     → bounding box
       005C     → tag # &1401
       0060     → 0006 (group object)
⑨ ③    0064     → 4C54 (length of group object)
       0068     → bounding box & name of group
       0084     → Draw objects to render backgnd
       ....
       4CB4     → hidden bytes inside tag
       4CB4     → 0000,1680,56833,6FB66
       4CC4     → 0001,7000026A,73A4,122948
       4CD4     → 0002,26A,122948,145124
       ....
       7354     → 26A,26A,08BAB43C,08BAB808
       7364     → 0007 (tag object)
⑩      7368     → 08BA44A4(length of tag)
       736C     → bounding box (all zero)
       737C     → tag # &1402
       7380     → 0006 (group object)
⑧ ④    7384     → 0024 (length of empty group)
       7388     → bounding box (all zero)
       7398     → name of group object (12 char)
  ⑤    73A4     → Draw file for page 1
  ⑥    122948   →Draw file for page 2
       ....
       08BAB43C →Draw file for last page
  ⑦    08BAB808 →End of 'Bound' file
```

A specific example of the internal format of a 'Bound' file. The part of the header at 000C identified by the word 'Bind' has been introduced so that loading just the first 40 bytes is enough to determine the memory requirements to load the background image and four pages of content.

such a file: if !RingBind is running and has a large enough Wimpslot for the 'Bound' file concerned, either to load it in full or just the parts necessary, it will accept the message and load and display the file.

If the message is ignored, then the !RingBind application will be executed and then has an opportunity to request enough memory to load the file in its entirity or, failing that, to load just the necessary parts. If there is still insufficient memory available it will abort with an error message.

Memory usage in version 0.10 will thus be confined to being able to load the first part of the 'Bound' file (from 0000 to ffff, about 30k depending how many 'tabs' are shown) as well as hold four pages in memory, space for the largest page (qqqq) is allocated for each, about 6Mbytes in total, in a continuous block so that two or four pages can be loaded in a single operation using OS_GBPB.

Creating a 'Bound' file

The first step is to create a directory full of Draw files, one for each page. This can be done using !PDF in a single 'save' operation.

The second step is to identify where each Chapter, Section and sub-section starts and ends, so that the three levels of 'move forwards' and 'move backwards' will work. Clicking the page corner will move two pages forward or back.

In the column to the left, the byte at 4CCB (&70) says that page 1 marks a Chapter, Section and sub-section break, with the three relevant bits set.

The generation programme 'ManGen9' requests the name of the directory where the Draw files are stored and expects to find a LIBRARY file there called 'ManualData' which will state the number of pages etc.

The individual pages are examined to determine the largest page width and height and the largest size of a single page.

The 'Bound' file (which renders as a Draw file showing just the background image) is thus created. For the RISC OS 5.28 User Guide this created a 'Bound' file of some 628 pages and about 140 Mbytes in size, with a maximum page size of 1236k.

To load just the background image and four of the pages of content would require a Wimpslot of just 10Mbytes. Version 0.06 of !RingBind simply loads the whole 'Bound' file into memory and thus requires a Wimpslot of about 150Mbytes to load the RISC OS 5.28 User Guide - this now seems a trifle excessive.

However the simple approach meant that I was quickly able to render two adjacent pages onto the background image using the simple redraw routine below. I could recognise a click over the bottom left or right hand corner of a page and display two pages further on or two pages back.

Right: *the adjustment to show an image of a right hand page being turned to the left as theta rises from 0 to PI/2.*

```
b=(1<<16)*0.1*zoom*SIN(theta)
LOCAL ERROR
ON ERROR LOCAL shift%=shift% OR 8
IF shift% AND 8 THEN b=0
!ddtran%=(1<<16)*zoom*COS(theta)
ddtran%!4=b
ddtran%!8=0
ddtran%!12=(1<<16)*zoom
SYS "DrawFile_Render",0,frm,end,ddtran%
RESTORE ERROR
```

Below: *the redraw code to show the background with left and right pages superimposed.*

```
1400 :   SYS "Wimp_RedrawWindow",,block% TO flag
1410 :   zoom=zoomL
1450 :   WHILE flag
1460 :     xlc%=block%!28-block%!4+block%!20
1470 :     ylc%=block%!32-block%!16+block%!24
1480 :     x2c%=block%!36-block%!4+block%!20
1490 :     y2c%=block%!40-block%!16+block%!24
1500 :     cx%=block%!4-block%!20
1510 :     cy%=block%!16-block%!24
1520 :     REM This specifies the window position on screen in OS units
1530 :     sx1%=block%!4:REM block%!4=screenXmin (L)
1540 :     sy1%=block%!8:REM block%!8=screenYmin (B)
1550 :     sx2%=block%!12:REM block%!12=screenXmax (R)
1560 :     sy2%=block%!16:REM block%!16=screenYmax (T)
1580 :     scx%=block%!20:REM block%!20=scrollX
1590 :     scy%=block%!24:REM block%!24=-scrollY
1730 :     duX%=ddReg%!24:
1740 :     duY%=ddReg%!36:
2640 :     REM Now render it  (page 2 on L, page 3 on R)
2650 :     !ddtran%=&10000*zoom
2660 :     ddtran%!4=0
2670 :     ddtran%!8=0
2680 :     ddtran%!12=&10000*zoom
2730 :     ddtran%!16=-(ddReg%!24)*zoom+256*cx%
2740 :     ddtran%!20=-(ddReg%!36)*zoom+256*cy%
2750 :     REM brown background
2760 :     SYS "DrawFile_Render",0,bufmd%,[hhhh]-bufmd%,ddtran%
2770 :     REM LH page
2780 :     ddtran%!16=-(ddReg%!24)*zoom+256*cx%+[mm]*zoom
2810 :     ddtran%!20=-(ddReg%!36)*zoom+256*cy%
2820 :     SYS "DrawFile_Render",0,bufmd%+[ptr2],[ptr3]-[ptr2],ddtran%
2840 :     ddtran%!16=-(ddReg%!24)*zoom+256*cx%+[pw]*zoom+[mm]*zoom
2850 :     ddtran%!20=-(ddReg%!36)*zoom+256*cy%
2860 :     SYS "DrawFile_Render",0,bufmd%+[ptr3],[ptr4]-[ptr3],ddtran%
2960 :     :
2970 :     SYS "Wimp_GetRectangle",,block% TO flag
2980 :   ENDWHILE
```

The background image is sized so that two pages from the PDF, side-by-side and touching, will have a small margin around their outside edges, except that there is extra space at the right hand edge for 'index tabs' that will jump to a specific page. Most (if not all) of the pages will have a blank border, i.e. the objects drawn on the page will not extend to the edge.

The bottom left- and right-hand corners of the page area on the background image will show a turned-up corner. This is inside a named Draw 'group' object (type=6). If the manual is shut, so that only its cover is in view, there will be no LH page and the draw object type for the bottom left will be altered to &4C0 (invisible named group object, another allocation from ROOL) so that the turned-up corner disappears.

The group objects containing the turned-up corners are named 'Fwd' and 'Back'. The group objects for each 'index tab' are named 'Tab_00nn' where '00nn' is the odd page number to jump to.

Within an individual Draw file for a particular page, a group object named 'PRef_00nn' would be treated as a 'hotlink' to page 00nn, if clicked upon.

It is easy to tell if the mouse is over a particular named group object by examining the bounding box of all group objects and finding the smallest box that encompasses the mouse position and then reading its name.

The group objects 'Fwd' and 'Back' simply move two pages forward or back when clicked upon.

Animation

It would be nice to display a page being turned - for the right hand page all I would need to do was to alter the first four elements of the transformation matrix to show a page being turned, its width shrinking and its right hand edge rising as it turned.

As it got to the vertical, the other side of the page would be seen, its width extending until it covered the view of the previous page.

The complicating factor here was that two screen banks would be required so that the Draw file data could be rendered to one bank while the other bank would respond to redraw requests. Swopping both banks and forcing a redraw would then display the updated view with the minimum of flickering.

Version 0.15 has improved error handling: most errors now restore a single screen bank before reporting the error so that the error window is visible. A mode change is recognised so that it knows memory for a second screen bank has been withdrawn - any animation is immediately terminated. Error trappping now detects errors generated by an attempt to plot a skew JPEG and plots it unskewed.

This version also uses a much smaller WimpSlot, approximately 12 Mbytes - enough to hold the background image and four of the largest page images. It therefore runs on VRPC and it will handle 'Bound' files larger than 512 Mbytes.

With version 0.15 is a copy of the programme to generate a 'Bound' file from a PDF, with full instructions. It is available from !Store along with the RISC OS 5.28 User Manual, which has now been released under the Apache licence, as a 'Bound' file.

An animated view of the page turning can be seen on my web site at: *https://www.svrsig.org/software/RingBind.htm*. This is somewhat slicker and faster than RISC OS can manage on older hardware (it has been captured by a series of screenshots and then combined into an animated gif file) but on ARMX6, Pi4 and Titanium it works at about that speed.

So to produce this piece of software I have had to request allocations from ROOL for application name (!RingBind), filetype (Bound, 1EB), draw object (4C0) and draw tags (&1401, &1402) all of which were granted very swiftly for which I am very grateful.

Points to note

The last draw object before the hidden bytes at eeee is a group object called 'Now' containing a text object which is a single 'bullet' character with a bounding box encompassing the whole slider bar but rendered at the appropriate point to mark the position within the document.

Rendering Draw file data using the DrawFile_Render SWI (as !RingBind does) can produce odd errors - for example '(Number)' if a font is not present or 'System variable SCSI$Path not found' if trying to plot a skewed JPEG. The latter error is trapped and resolved.

Creating a PDF using the Postscript level 3 drivers will have all fonts embedded as type-1 fonts rather than as lower quality, named, type-3 fonts with font mapping to specify names for other operating systems. Use only Corpus, Homerton and Trinity fonts so that !PDF will render them correctly and output Draw files that will render on RISC OS. This also applied to fonts used within images in the document.

!FontFX can be used to convert text in non-standard fonts to paths.

Chris Hall *chris@svrsig.org*