Introduction

This document describes a method of including sound and embedded pane animation in a Help file.

It is assumed that you will generate your help files using EditHelp available from **Analogue Information Systems Ltd., 1 Warrender Park Crescent, Edinburgh EH9 1DX, (031 228 6008)**. EditHelp is shareware and is available free to anyone who sends a formatted floppy and a stamped-addressed envelope. Shareware registration is £25+VAT ($40+postage outside the E.C.).

Sound production uses the standard Windows Multimedia Interface. You will need either

- a sound card and compatible driver

- a driver for the PC internal speaker

The system calls for sound production are executed via the DLL:

PlaySnd.DLL

The animation is executed via the DLL:

AnimHelp.DLL

AnimHelp is under development. This doument and the specification of AnimHelp should be treated as "provisional". I'll do my best to ensure that backward compatibility is maintained when the first "real" version is released.

<u>**Sound Production**</u>

The PlaySnd DLL allows you to play a WAV file from within a help file.

PlaySnd.DLL contains two functions:

PLAYSND:	plays a WAV file that has been stored in the HLP file as "baggage"
PLAYFILE:	plays a WAV file that is held on disk

PLAYSND function

The EditHelp DLL call

```
{dll=PLAYSND.dll,PLAYSND(S:qchPath,S:"sine.wav",u:0),play sound}
```

plays the file SINE.WAV where SINE.WAV has been stored in this HLP file.

(For those of you not using EditHelp, EditHelp translated this DLL call into the RTF:

```
{\f0\fs20\cf0\uldb play sound}
{\v !RegisterRoutine("PLAYSND.dll","PLAYSND","SSu");
  PLAYSND(qchPath,"sine.wav",0)}
```

which registers then calls the DLL.)

Whenever the function is executed, it will play the WAV file once.

PLAYSND takes two string and one unsigned integer parameter:

HelpFile:	the first parameter is the name of the help file which contains the sound. You can use the Predefined Variable qchPath which specifies the help file containing the current topic
SoundFile:	the second parameter is the name of the sound file contained in the help file
Flags:	sum of:

1:	play asynchronously
8:	loop the sound until next PLAYSND
16:	don't stop any currently playing sound

If "play asynchronously" is specified then the player will start the sound and immediately continue on with the next command. Otherwise, the player will wait until the sound has finished before continuing.

EditHelp will include a file inside a help file (HLP) if the directory containing the source of your help file (EDH) also contains the file

```
BAGGAGE.EPJ
```

BAGGAGE.EPJ is a text file. Each line is the name of a file to be copied into the HLP file as "Baggage". For instance, if the BAGGAGE.EPJ file contained the text:

```
sine.wav
```

then the "sine.wav" file would be included in the HLP file.

(Those of you not using EditHelp should include the line
sine.wav
in the BAGGAGE section of the HPJ file.)

A help file contains its own file system. Most of the files are used by WinHelp or whatever browser you are using to display the help file. Other "internal files" included as bagge can be read by other programs or DLLs.

PLAYFILE function

The DLL call

```
   {dll=PLAYSND.dll,PLAYFILE(S:"sine.wav",u:0),play file}
```

has the same function as a call to PLAYSND() but it reads its source from an extrnal file - not the baggage in the HLP file.

PLAYFILE takes one string string and one unsigned integer parameter:

SoundFile: the name of the sound file

Flags: sum of:

1:	play asynchronously
8:	loop the sound until next PLAYSND
16:	don't stop any currently playing sound

External files are easier to alter when you are creating your HLP file. But I think it looks neater if the final product is a single HLP file. If you release your product as several files, it's easier for one or two of them to get lost.

<u>**Simple Animation**</u>

Animation in an embedded pane can be produced by calling the HelpAnim DLL. For instance, the EditHelp embed call:

```
{embed=HelpAnim.dll,AnimWnd,1/dino.txt}\
```

creates an animation pane in the HLP file which is controlled by the file Dino.TXT.

In this call:

- "AnimWnd" is the Window Class of the animation pane

- "1" is the id-string of the pane in ths topic (in case the topic contains two panes with the same class)

- "dino.txt" is the animation control file.

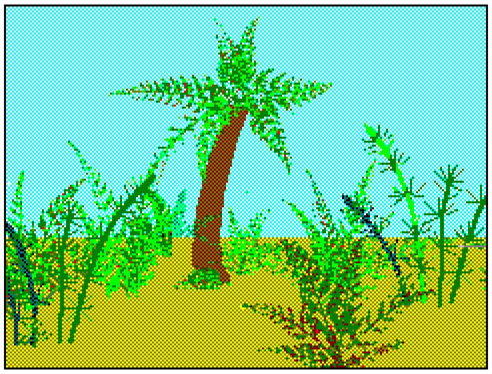
An Embedded pane is a rectangular region of the Help "page" which can contain graphics, animation, simulation, buttons, editable text fields or whatever features the DLL provides.

(For those of you not using EditHelp, EditHelp translated the embed command above into the RTF:
{ewc HelpAnim.dll,AnimWnd,1/dino.txt}
which registers the Window Class "AnimWnd" with the author string "1/dino.txt".)

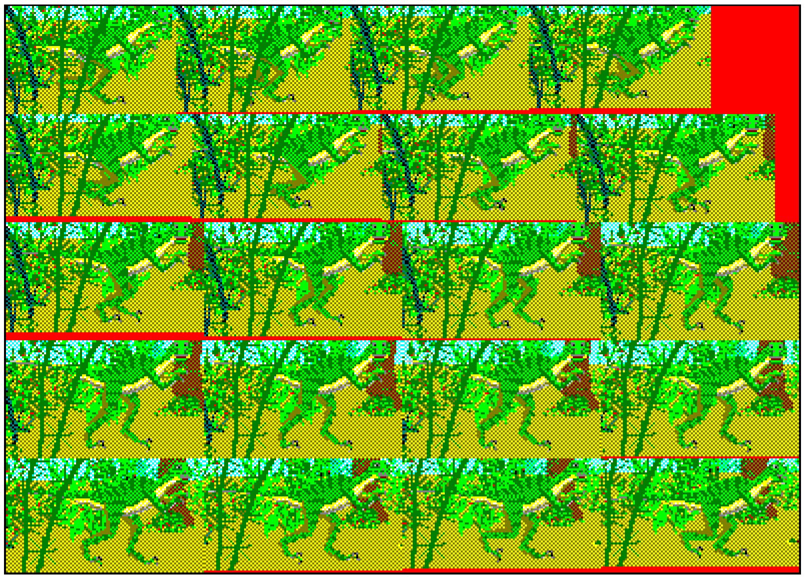
The implementation uses "dirty rectangle" animation: only the parts of the screen which have changed are stored as bitmaps.

The source bitmaps and the playing instructions are kept in the HLP file as "Baggage". They are not compressed - which is why the HLP file is so big and takes so long to load. A compressed version should be available soon.

Typical source bitmaps would be a background:



and a sequence of rectangles to be pasted over the background:



At present, there is no animation authoring tool. You have to calculate the "dirty rectangles" by hand. An authoring tool is planned.

The embed command contains the name of a file which specifies how the rectangles should be copied over the background. For instance in

```
{embed=HelpAnim.dll,AnimWnd,1/dino.txt}\
```

the file "Dino.txt" contains the animation instructions. "Dino.txt" must have been included in the HLP as Baggage.

"Dino.txt" contains:

```
2                          version
8                          number of bmp files
bmp1.bmp                   first bmp file
bmp2.bmp                   second bmp file
      :                                          :
      :                                          :
bmp8.bmp                   Nth bmp file
0                          number of sound files
354                   number of command lines
SB 1                  size of pane = size of first bitmap
BB 1                  background bitmap = 1
WS 50                 wait N ms
LB 1                  label = 1
BD 1 0 0                   draw bitmap 1 to (0,0)
BR 0 106 1 27 0 0 2   copy rectange from Bitmap 2
WS 50                 wait 50mS
BR 0 105 1 28 1 0 2   copy rectange from Bitmap 2
WS 50                 wait 50mS
      :
      :
BR 230 105 11 57 13 59 8 copy rectange from Bitmap 8
WS 50                 wait 50mS
BR 232 105 9 57 24 59 8  copy rectange from Bitmap 8
```

```
WS 50                   wait 50mS
GT 1                    goto label 1
```

The animation instructions are a sort of program. Each line is executed in turn and you can define gotos and labels.

Each line is a two letter command code followed by up to seven numbers separated by blanks. The following commands are available:

BR x_dest y_dest width height x_source y_source bitmapNum
copy a rectangle from a bitmap to the pane

SZ wWin hWin
the size of the pane

SZ bitmapNum
the size of the pane is equal to the saize of the bitmap

LB label
set label

GT label
goto label

CG label
on mouse click, goto label

CR x1 y1 x2 y2 label
on mouse click in rectangle (x1,y1,x2,y2), goto label

WP
pause until user clicks mouse

WS mSec
pause for N mSec

BC Red Green Blue
background colour

BB bitmapNum
set background to bitmap

BD bitmapNum x_dest y_dest
draw a whole bitmap

PS soundFileNum flags
play a sound file (for "flags" see PLAYSND above)

GS label
gosub label

RT
return from gosub

AS variable value
assign value to variable

This list will be extended from time to time - why not write and say what you think is essential.

Notes:

- Bitmaps (and sound files) are refered to by number: bitmap 1 is the first bitmap in the list of bitmaps
- The bitmaps and WAV files should be included in the HLP file as Baggage.
- When the commands are executed, the first command (optionally following a goto and label) must be eithr SZ or SB to set the size of the pane.
- On a WS command, the player will pause for a certain number of clock ticks; the system clock ticks every 55mS so the delay is rounded to the nearest 55mS.

Twenty six variables are supported named 'A' to 'Z'. You can use a variable anywhere that you can use a number; except that in the **AS** (ASsign) command, the first parameter must be the name (letter) of the variable which will receive the value.

So, if you look back at the "Dino.txt" example above, you'll see that:

- it defines 8 bitmaps and no sound files
- then it sets the size of the pane to the size of bitmap 1
- then it sets bitmap 1 to be the background
- then it goes into a loop, every 50 mS it copies a different rectangle from bitmaps 2 to 8

When the frames are shown in sequence, only the "BR" rectangles are drawn for each frame. So it is important that each "BR" rectangle is big enough to overwrite the previous one.

The "Dino.txt" commands been designed so that the player can show the "frames" in any order. You can add buttons to single-step the animation forwards and backwards (see below). The commands define a background bitmap; when a frame is to be shown out of sequence, the player draws the background bitmap followed by all the "BR" rectangles up to the next "WP" or "WS".

External and compiled files

Normally, bitmaps and WAV files should be included in the HLP file as Baggage. However, for debugging, you can precede the filename with a "-" character: the bitmap or WAV file will then be read from an external file.

Similarly, in the **embed** command, if the filename is preceded by a "-" character then the command file and the bitmaps (but not, at the moment, the WAV files) will then be read from an external file. For instance:

```
{embed=HelpAnim.dll,AnimWnd,1/-dino.txt}\
```

Reading a command file like "dino.txt" takes a long time. It's much faster if the file has been "compiled" to a file of bytes. To compile the file, call the **embed** command with the filename preceded by a "+" character. The HelpAnim DLL will read the file as an external file and write a "compiled" version with a ".DAT' extension - for instance

```
dino.dat
```

When you next use the **embed** command, you can use the ".DAT" file as a baggage file - it will load a lot faster. You should never try to use a ".DAT" file with a "-" or "+" as an external file: it won't work.

More Complex Animation

The "PicPos.txt" file controls a more complex example of animation. (It shows the "FM Encoding" topic in the HelpAnim.HLP sample file.)

"PicPos.txt" contains the following commands:

```
1                               version
4                               number of bitmaps
bg.bmp                    bitmap 1
bits.bmp                  bitmap 2
disk.bmp                        bitmap 3
text.bmp                  bitmap 4
1                               number of sound files
sine.wav                  sound file 1
323                       number of command lines
SB 1                      size of pane
LB 1                      label 1
BB 1                      background bitmap
BD 1 0 0                        draw bitmap 1
BR   4   4 315 50 0   0  4      draw text message
CG 4                      on click goto 4
BR 470 32 55 34 109 120 2       draw bitmap 2
GT 20                     goto 20
LB 4                      label 1
BR 377   2  70 10 0 118  2      copy "data bits" bitmap
WS 150                    pause
        :
BR 377   2  70 10 377   2  1    copy "data bits" bitmap
WS 150                    pause
CG 2                      on click goto 2
GT 20                     goto 20
LB 2                      label 2
        :
        :                       "encode data"
        :
        :                       "write data"
        :
GT 1                      start again
LB 20                     label 20
BR  82 140 233 123   0    0  3  draw "disk"
WS 50                     pause
BR  82 140 233 123   0  123  3  draw "disk" rotated
WS 50                     pause
BR  82 140 233 123 233    0  3  draw "disk" rotated
WS 50                     pause
BR  82 140 233 123 233  123  3  draw "disk" rotated
WS 50                     pause
GT 20                     goto 20
```

If you look at the sample help topic, you'll see that the "disk" appears to rotate. That is controlled by the loop "LB 20" to "GT 20" which repeated draws four views of the "rotating bits". So long as nothing else is happening, the player executes the loop.

The "CG" commands specify that when the user clicks the mouse, the player should jump to a different label. In the example above, the first time the user clicks, the player jumps to label "LB 1", the second time they click, to label "LB 2", etc.

User Control over the animation

The **CR** command allows you to associate an area of the window with a lable. If the user clicks within the rectangle, the program jumps to the label:

```
CR  x1  y1  x2  y2  label
```

The "gears" example shows an example of the use of the **CR** command.

Control Buttons

WinHelp - the standard viewer used with HLP files - does not support a good keyboard interface to embedded panes. It's much better to control the panes from DLL calls in separate text or bitmap hotspots.

ANIMFUNC

The "dinosaur" topic in the HelpAnim.HLP sample file contains four buttons. Each button calls the ANIMFUNC function in the HelpAnim DLL. For instance:

{dll=HelpAnim.dll,ANIMFUNC(S:"AnimWnd",S:"1",i:1),bmp=btn2.bmp}

The AnimFunc function takes three parameters:

- **S:"AnimWnd"** is the Window Class of the animation pane

- **S:"1"** is the id-string of the pane

- **i:1** is the command to be sent to the pane

S:"AnimWnd" and **S:"1"** correspond to the similar parameters in the {embed=...} command. The third parameter specifies what the animation player should do:

0:	stop the animation at the next "WP" or "WP" command and wait for another command
1:	run the animation forward at normal speed
2:	backspace the animation by one "frame" and stop
3:	move the animation forward by one "frame" and stop
4:	run the animation forward at high speed

The player treats all the commands between one "WP" (or "WS") and the next as a "frame". When the animation is run at high speed, all the "WS" pauses are ignored.

If there is no outstanding "CG" command (or "CG 0" is outstanding), then when the user clicks on the pane, the animation toggles between running forward at normal speed and stopped.

If there is (non-zero) "CG" command outstanding, then a

{dll= ... ,ANIMFUNC(... ,i:1), ... }

command is treated as though the user clicked the mouse on the pane and the player jumps to the "CG" label. If you are using "CG" commands, you should always include a DLL call to ANIMFUNC somewhere in the topic so that your animation can be controlled by users without a mouse.

ANIMEXEC

The Gears.HLP sample file contains several text buttons. Each button calls the ANIMEXEC function in the HelpAnim DLL. For instance:

{dll=HelpAnim.dll,ANIMEXEC(S:"AnimWnd",S:"1",S:"GT 600"),neutral}

The ANIMEXEC function takes three parameters:

- **S:"AnimWnd"** is the Window Class of the animation pane

- **S:"1"** is the id-string of the pane

- **S:"GT 600"** is a HelpAnim command

S:"AnimWnd" and **S:"1"** correspond to the similar parameters in the {embed=...} command.

The third parameter can be any HelpAnim command. In the Gears.HLP example, it is a jump to a label to within the program.