Technical Queries

Alan Wrigley answers more of your questions.

Dear Sir
I would like to display text in a larger than normal size within a window, but I can't see a way to do it unless I use an outline font or draw the characters myself. Is this correct?

Colin Murray

The system font when used in VDU 5 mode *can* in fact be scaled on screen on both the x and y axes, using the multi-purpose VDU 23 command. The full syntax for this particular operation is as follows:

```
VDU 23,17,7,flags,dx;dy;0;0;
```

Bits 1 and 2 of the flags byte are used to indicate whether the character size or spacing or both are to be affected. If bit 1 is set, then *dx* and *dy* are used to set the horizontal and vertical size respectively of any character subsequently displayed on the screen, in pixels. If bit 2 is set, *dx* and *dy* set the character spacing in the same way. The normal character size is 8x8 pixels (in mode 12 for example), so to set both the size and spacing to double on each axis, you would use:

```
VDU 23,17,7,6,16;16;0;0;
```

This is not the end of the story, however, since the character pixel size is dependent on mode. If you are using a multisync mode, for example mode 20, then in order that normal text should look the same size on the screen as with a non-multisync mode, the vertical character size and spacing are doubled to 16 pixels. So to make your program mode independent (which all Wimp programs should be), you must first ascertain the character size and then scale that in the VDU command, rather than scaling an absolute value of 8x8. To do this, you need to call OS_ReadVduVariables to read variable numbers 162-165, which are x-size, y-size, x-spacing and y-spacing respectively for VDU 5 characters. This call requires an input buffer to be passed in R0, in which the variable numbers to be read are placed in consecutive words, terminated by -1, and an output buffer to be passed in R1, into which the values returned are placed in the corresponding positions. So to double up all the values as before, you would use the following code:

```
DIM in% 19,out% 15
!in%=162:in%!4=163
in%!8=164:in%!12=165:in%!16=-1
SYS "OS_ReadVduVariables",in%,out%
xsize%=!out%:ysize%=out%!4
xspace%=out%!8:yspace%=out%!12
VDU 23,17,7,2,xsize%*2;ysize%*2;0;0;
VDU 23,17,7,4,xspace%*2;yspace%*2;0;0;
```

Note that in this case we have issued two separate commands to set first the size and then the spacing. This is to cater for the possibility that the size and spacing may not have the same value. Note, too, that once you have issued these VDU commands, *all* character output to the screen will be at the new size, regardless of which application is outputting the characters. For this reason you must revert to normal size characters as soon as you have finished redrawing your window, by repeating the VDU commands, this time using the values obtained for *xsize%* etc. above.

Dear Sir
When converting a variable into a string using STR$ there is often a long mantissa. What is the easiest way to control the accuracy of the conversion?

Keith Vernon

Technical Queries

While it is easy for a computer to represent integer values, since these can be based simply on a bit count, it is not so easy to represent floating point values with absolute accuracy. Whatever algorithm is used to encode the number, there will inevitably be rounding errors in certain cases. You can see this at first hand by typing in the following line from Basic:

```
FOR i=0 TO 10 STEP 0.1:PRINT i:NEXT
```

At some point you will see numbers such as 3.59999999 begin to appear. Further rounding errors occur when you use STR$, as you have pointed out. Replace *PRINT i* in the above line with *PRINT STR$i*, and you will see a different set of figures on the screen.

As in so many cases, however, Basic comes to the rescue, this time with the special variable @% which is specifically provided to determine the format in which numbers are to be displayed. Byte 0 of this variable gives the field width when tabulating with commas. By default this is 10, so if you type:

```
PRINT 1,2,3,4
```

you will see that the numbers are spaced out by 10 columns each.

Bytes 1 and 2 of the variable determine the number of digits printed, and the type of format to be used respectively. If fixed format is chosen (byte 2=2), byte 1 represents the number of digits after the decimal point, while in general format (byte 2=0), byte 1 represents the total number of digits. The default is general format with 9 digits. The number of digits displayed affects the accuracy of the number; a greater number of decimal places is more likely to introduce the rounding errors we have described above.

For example, if you alter the digits to 8 instead of 9 before typing in the line given above, as follows:

```
@%=&80A
FOR i=0 TO 10 STEP 0.1:PRINT i:NEXT
```

you will notice a marked difference.

However, if you then substitute *PRINT i* with *PRINT STR$i* as before, the rounding errors will still be there. To cure this, we need to set byte 3 of @% to a non-zero value, which indicates that the format set with the rest of the variable should also be used by STR$. For example:

```
@%=&102040A
```

would set a field width of 10 and print all figures to 4 decimal places, employing the same format if STR$ is used. If you set the format to suit your own particular requirements, this should effectively overcome the rounding problems that occur with the default settings.

In some cases, it may be easiest to forget about floating point numbers altogether and work with integers on multiples of 10 or 100 times the actual value. For example, suppose you are writing a program that handles monetary values. Provided that you will not be handling fractions of a penny, you could easily do all your internal calculations in pence, using integer variables, and only introduce decimal points into the actual string displayed on the screen, not into the values themselves. To do this, you could use a function such as:

```
DEFFNstring(val%)
a%=ABS(val%):a$=STR$(a%)
IF a%<100 a$="0"+a$:IF a%<10 a$="0"+a$
IF val%<0 a$="-"+a$
=LEFT$(a$,LEN(a$)-2)+"."+RIGHT$(a$,2)
```

which will turn a positive or negative

value passed to it in pence (e.g. 1234 or -12) into a string in pounds and pence (12.34, -0.12).

MORE ON SPRITE ICONS

Technical Queries in the April issue (Volume 5 Issue 5) addressed the question of how to display sprite icons in windows created using FormEd. Richard Hallas has written to say that the methods proposed in the article were more drastic than was necessary, since there is a word in the window definition itself specifically provided for a sprite area pointer. This is at block%+64, and so all that is required is to insert the following line between loading the template and creating the window:

```
block%!64=spritearea%
```

However, this will only work as it stands for text-plus-sprite icons; any sprite-only icons still need to have their own definitions altered to point to the sprite area. The word which needs to be altered in this case for each relevant icon is block%+88+32*icon+24. Also, if you have set the global pointer in the window definition as described above, you cannot then mix sprites from your own area and from the Wimp pool without also altering the icon definitions of one group.