Autosize

Purpose: To show how to programmatically position your controls at run time so that the controls and their surrounding form remain proportionately the same size regardless of the display resolution or the dots per inch of the display driver's screen fonts.

Background: After completing my first VB app, I was all jazzed up to show it off to my brother in St. Louis last Christmas. Low and behold, when the app came up on his screen, it looked all screwed up. Many of the controls which I had visually "drawn" on the form were nowhere close to where they had been placed in design mode on my development machine. This was an embarrassing way to learn about the fact that Visual Basic V1.0 is NOT screen device independent.

This means one thing. If you want to write windows applications which will look proportionately the same regardless of the display driver, then you must size and position all of your controls and forms in code. This may sound like a P.I.T.A., but once you get used to it, it's not so bad.

What's the deal, anyway? I'm no expert on this subject, but I think I've got the gist of it. So I'll try to help you understand what the deal is with resolution and DPI (Dots per Inch).

Resolution: As resolution increases, the windows desktop will get larger. That is, it will be like you are zooming out from the desktop. All Items on the desktop get smaller with increased resolution and hence you can fit more stuff on your screen. If all we had to deal with was resolution, then screen device independence would be a non-issue. This is because the number of twips per pixel would remain constant at 15.

DPI (Dots per logical Inch): The thing which screws everything up is when different display drivers map the screen fonts at more or less Dots per logical inch. This is often the case when 120 DPI (Large font) drivers are used at runtime when 96 DPI drivers were used for development. 120 DPI screen drivers are often used to make a 10 point font big enough to read without a magnifying glass in 8514-A mode. Consequently, you gain clarity at the expense of a smaller desktop. Also, some Super-VGA drivers are mapped to 120 DPI, instead of 96 DPI. The result is that a Super-VGA, 120 DPI windows desktop is **about the same size** as a regular VGA desktop displayed at 96 DPI! If you recently switched to Super VGA and noticed no increase in desktop real estate, then you no doubt installed a 120 DPI driver instead of 96.

If all display drivers used screen fonts which were 96 DPI (Like std VGA), then a control which is "drawn" in VB as 1440 Twips wide (one logical inch) would be 96 pixels wide on all systems. Likewise, a control "drawn" 360 Twips (1/4 inch) below the first control would always be in the same relative position at various resolutions if the screen fonts stayed at 96 DPI. If you go to 120 DPI though, the twips per pixel ratio has changed to 12...the same font is now "Bigger", and your controls have all fattened up to make way for the font. The fatter controls make the distance between controls get smaller, so your controls appear more bunched up than you drew them. They also appear "Shorter" in relation to the words they contain, and may not hold all of the letters necessary to display the whole data item.

How Autosize works: This program bases all its control and form sizing and placement on the size of a standard font in pixels. The standard font chosen is the default font used by command buttons, text boxes and list boxes. Usually this is Helv 8.25. This makes perfect sense, because most of our controls display information as text. A command button, for example, should be about twice as high as the letters displayed in it, and wide enough to display the caption adequately. If our sizing scheme is based on the size of the button's font, then no matter what size (in pixels) the display driver shows in the button, the button's dimensions will look proportionately the same. The same goes for all the other controls which contain text. A listbox which defined as ten characters wide will always be ten characters wide if we size it based on the font size which it

contains (in pixels).

Picture boxes are not covered here because there are so many ways to scale both the picture boxes and their contents.

Use Scalemode = 3 (Pixels): Pixel scalemode (3) is always used as the default for forms. This makes it easy to do API calls to the GDI, because all these calls expect drawing parameters in pixels. Also, the measurement which is affected by different drivers is the Twip to PIXEL ratio.

How differing DPI display drivers change the number of Twips per Pixel:

Standard Display driver Dots per logical inch:

	CGA	**EGA**	**VGA**	**S-VGA**	**8514-A**
X	96	96	96	96 or 120	120
Y	48	72	96	96 or 120	120

Resulting Twips per Pixel:

	CGA	EGA	VGA	S-VGA	8514-A
X	15	15	15	15 or 12	12
Y	30	20	15	15 or 12	12

Autosize calls two procedures to get its work done - Init_Measures and MakeForm

Init_Measures - gets the horizontal and vertical Twips Per Pixel. It alse sets the scale mode to Pixels and defines our Standard units of measurement called gHStd and gVStd.

MakeForm - This is where the dirty work is done. Starting with the control located in the upper left of the form, and working its way to the lower right, Makeform sizes and positions each control in relation to either the edge of the form, or the control placed before it. At the end of the procedure, Makeform sizes the form itself and centers it in the screen (slightly high).

DrawForm - You may notice that I drew all my labels directly on the form. This conserves precious resources which would have been wasted using label controls. Eliminating a bunch of label controls also speeds up the form's loading time. I Avoid label controls like the plague.

I hope this example is able to help you avoid the countless hous it took me to get this screen device independence thing straightened out.

Sincerely,

Bob Scherer
Newport Beach, CA

CIS number: 76237,514