

Re: Mike: Restating your explanation

Source:

<http://www.tech-archive.net/Archive/VB/microsoft.public.vb.general.discussion/2006-07/msg00902.html>

- *From:* Randy Gardner <RandyGardner@xxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Wed, 12 Jul 2006 08:02:02 -0700
-

Hi Mike:

I understand all of your explanation. Thank you!

Wrongly, I assumed the dw variable type was appropriate. I now see my error, dw (DoubleWord). The results I was getting should have tipped me off too, but, I guess my frustration got the better of me.

I worked through the issue of logical and physical measurements on monitors when I remembered that pixels on monitors(dot pitch) come in different sizes, or setting as you explained, i.e., the error. Printers on the other hand are very high resolution (600x600, per square in.) and even better on some models so they can get the physical measurement right.

If I understand the basics of screen size and display then the following deduction for my situation should be correct:

I need to display real world values of: X (0 to +40) and Y (-2 to + 200) and the units are inches (x and/or y) and force (y). I can't get that done with any of the real world settings without scaling the data to fit.

As an example: I could set the display to Scalemode = vbInches and in software scale my data to fit the physical limits of the display when the max value exceeds the limits of the display which is what I have done in the past.

From my reading and your code example: ScaleMode = vbUser should allow me to

make a scale for any physical display size that has limits that are the worst case values of my data. Simply stated: my graphical data should always use the entire display regardless of the extremes of the worst case values (x= -10k or +10K, y = -.01 or +10M).

If that is all accurate then we have arrived at my problem.

Re: Mike: Restating your explanation

More by accident and definitely without total understanding I have been able to get an origin where I wanted it, but I have almost always scaled the data to fit.

In Fact, another MSDN good Samaritan like yourself gave me some code that I just used without a whole lot of understanding. I didn't have a problem because the values were small enough to fit the physical display.

1. I was able to effect the origin and scale using: (where CSFullWidth and CSFullHeight limits of my data)

$$X1 = -CSFullWidth / 2$$
$$X2 = CSFullWidth / 2$$
$$Y1 = CSFullHeight / 2$$
$$Y2 = -CSFullHeight / 2$$

Object.Scale (X1, Y1)-(X2, Y2)

The hardcoded 2 could also be a variable the represented a ratio (-:+). I also zoomed the display by changing the height and width values with a coefficient.

When I tried to reuse the same code in another program I was getting strange results. That is when I decided it was time to learn rather than borrow.

I still don't see why the above code should not work. The scale method sets the scalemode to vbUser and it should put the origin proportional to the X and Y values and the X and Y values should set the display limits at my min/max data values.(?)

I'm going to play with it somemore, may be I did something stupid!

If you are willing, I really believe the code example I asked for would help me immensely, especially if the option code used the Scale(x1,y1)-(x2,y2) method which is the alternative to directly specifying object.ScaleWidth/Height.

I really appreciate your help!!

—

Thank you,

Randy

"Mike Williams" wrote:

Re: Mike: Restating your explanation

"Randy Gardner" <RandyGardner@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message
news:23232E02-A654-4AC6-8299-3AD7831AD90D@xxxxxxxxxxxxxxxxxxxx

```
Hi Mike: I have been studing hard, still confused! <big snip>
I understand 90% of your code. I believe my problem is how
VB deals with scaling, the term "units"
The following is and example of a portion of my confusion:
dw = Picture1.ScaleX(1, vbInches, vbTwips)
dw = 1440 " Right"
dw = Picture1.ScaleX(1, vbInches, vbMillimeters)
dw = 25 " almost Right"
dw = Picture1.ScaleX(1, vbMillimeters, vbInches)
dw = 0 " Wrong" = .03937"
```

The reason you're getting incorrect answers in the above code is simply because you are getting the result into the variable dw, which in my code example was declared as a Long. Longs of course can hold only integers (whole numbers). If you change the declaration to [Dim dw as Single] then all the above lines of code will produce the correct result. The reason I used a Long in my own code is simply that I was calculating a value to use to set the DrawWidth. If you check the VB help files you'll see that DrawWidth is expressed in pixels, regardless of the ScaleMode you are using. Pixels are the smallest unit on the device (screen or printer or whatever). It isn't possible to have fractions of a pixel (apart from in some esoteric and rarely used cases which I won't go into here). Therefore a Long is a suitable variable type when you are converting other units to pixels, but for general purpose conversions between different units of measurement you need to use a Single.

. . . and especially VB actual unit of measure(Twips ???)
for drawing on forms, printers, etc.

As you're probably already aware, Visual Basic uses eight different ScaleModes. Of these, five are "real world units" (that is to say, they are units of measurement commonly used in the real world) and these five units of measurement have a fixed size relationship, each to the other. These units are:

- Inches
- Points (there are 72 points in one inch)
- Twips (there are 20 twips in one point)
- Centimeters (there are 2.54 centimeters in one inch)
- Millimeters (there are 10 millimeters in one centimeter)

On a computer, and in the real world, those five units of measurement always bear the relationship to each other as shown above. Twips are nothing special. There are always 1440 twips in an inch, just as there are always

Re: Mike: Restating your explanation

2.54 centimeters in an inch. (By the way, before someone pulls me up on this, there have throughout the years been all sorts of different definitions of a twip, and different definitions of a point and of an inch for that matter, but the twips used by Windows and by VB and by almost everyone else in the world these days have 72 of them in a point and 1440 of them in an inch).

At this point I think I should make it clear that as far as Windows is concerned (the screen display) an inch is a "logical inch" and it is not necessarily the same size as a "real world inch". In other words, if you set a Form's ScaleMode to vbInches and draw a rectangle that is one inch wide it will not necessarily be the same size as an inch on a ruler that you might hold up against the screen. On some computers it will be smaller than a real world inch and on others it will be larger than a real world inch and on a few machines it will be exactly the same size. This doesn't matter though for most purposes. The important thing is that all five "computer measuring units" bear a fixed relationship to each other, and that relationship is exactly the same as the relationship between those five measuring units in the real world. Printers are different. On a printer a "logical inch" is exactly the same size as a "real world inch".

Now we come to pixels. A computer display, as I'm sure you are aware, is essentially a large matrix of pixels. When the system draws something on the display it has to "light up" specific pixels in specific colours. In order to know what to do, the computer needs to be told how many pixels it needs to light up to draw a line of pixels that is one logical inch wide. The Windows operating systems tells the computer how many pixels there are in a logical inch. It can be different on different machines, and is actually user definable (it used to be almost totally user definable in Win98, but WinXP gives the user less choice in the matter). On most systems there are 96 pixels in a logical inch, and on some systems there are 120 pixels in a logical inch. Other systems can have different settings. Windows (and VB) generally look after all this stuff for you. For example, if you set a Form's ScaleMode to vbInches and draw a line one unit wide (one inch) then VB will check the operating system to see how many pixels there are in a logical inch and it will draw the required number of pixels for you (typically 96 on most machines). However, if you're using something that requires you to specify its size in pixels (such as setting the DrawWidth property of a Picture Box) and if you want the DrawWidth to be a specific thickness in inches then you need to perform the conversion yourself. The ScaleX and the ScaleY methods will convert between different units for you (one is used for measurements in the horizontal direction and the other for measurements in the vertical direction, but they usually both return the same result except on certain quite unusual systems). The ScaleX and ScaleY methods need to be told which device to check (because a screen will have a different number of pixels per logical inch than a printer) so you always need to specify the device you are interested in. For example, if you want to know how many screen pixels there are in a logical inch on your own machine you could use:

Dim p As Single

Re: Mike: Restating your explanation

```
p = Picture1.ScaleX(1, vbInches, vbPixels)
MsgBox p
```

Printers are similar in that a printer also needs to know how many "printer pixels" there are in a logical inch, and to check this you could use:

```
Dim p As Single
p = Printer.ScaleX(1, vbInches, vbPixels)
MsgBox p
```

One little problem with the above code for the printer is that on many systems it will produce a runtime error when run on its own, because it attempts to perform a calculation in respect of a printer that might not yet have been initialised, so it is wise to "give the printer a little kick" to wake it up first. This is usually performed by a Printer.Print statement (which you should always start all print jobs with), but you can also do it in various other ways that will not cause a print job to be started:

```
Printer.Orientation = Printer.Orientation
Dim p As Single
p = Printer.ScaleX(1, vbInches, vbPixels)
MsgBox p
```

I think it would be wise of you to make sure you fully understand all of the above before delving any deeper into this stuff, so I think I'll leave it at that for now. Post again if you have any questions.

Mike