# Stephen's Personal Blog

Everything's Programming

« TD: Why Did A California Court Hide All Of WikiLeaks Over A Single Document
IE Developer Toolbar »

## How to Calculate the Character Width Accross Fonts and Points

Ok, this is going to be long and technical. I'm writing it up for all the fools who, like me, have to figure how many pixels wide an Input field on a web page should be when the font and size are variable. Instead of letting the browser figure out the size.

This was a hideous problem to resolve, because it seems seductively simple at first. Just multiply a constant by the number of characters in a field. Give that a shot. The code I was replacing use 8 px/char on a Verdana 8pt. It doesn't work. Short fields are too narrow and long fields are massively wider than necessary. And this strategy doesn't account for other fonts the user can setup like Courier New for pt size from 6-18.

Each font has a unique character width. Each pt size has a unique character width.

My solution has a few parameters. The exact string is unknown. Font can be anything that the user can type and the system recognizes. Pt size again can be anything. Width is in pixels set on the Style attribute of an HTML Input tag rendered in Internet Explorer.

The system has a few defaults and I have a few favorites. So, I selected Arial, Consolas, Courier New, Georgia, Lucida Console, MS Sans Serif, Tahoma, Times New Roman, Trebuchet, and Verdana. On Point sizes ranging from 6-18, but preferably this is a function which can handle any Pt size.

Now, we see Windows, Word, and Internet Explorer solve this problem every day. The text box looks the "right" size. So, this problem has a good solution, because we see it and don't even think about it. Perhaps the operation system will have an API or .Net call that will help us figure it our or do the calculation for us.

It does. And it's called TEXTMETRIC In .Net 2.0 and earlier you have to reference GDI32.dll and call the API. .Net 3.0+ has a FormattedText class that seems to do exactly what we want. Unfortunately, this project is in .Net 2.0 and FormattedText is unavailable. There is yet one more way that will work in all the .Net versions. Make a Graphics object and use MeasureString to find the pixel width of a specific string.

So, lets take these one at a time. TextMetric can give you lots of information about a font. Of particular interest are MaxCharWidth and AveCharWidth. There are many ancillary measurements.

I ran a test program that would spit all of the Max and Ave(Average) character widths to a file. There is a large amount of variation here. For example, Arial 8pt has a Max of 29 and an Ave of 5. I thought I could make a functions that weighted the Max and Ave and generate a good result. This proved difficult and did not carry over to other fonts and other pt sizes. Each one would have to be guesstimated and the result might still need tweaking.

A lot of time was spent analyzing the differences between Pt sizes within one font. When you plot Max and Ave out, the graph is pretty close to a straight line. There's some obvious rounding that causes a bit of stair stepping, but it's not too bad.

What really sucked was the multiple characters. I had assumed that character width, on average, is constant. Yes, I know most of these are non-proportional fonts, but overall it should level out. Wrong!

This solution was not going to work. And analyzing the Max and Ave font sizes wasn't revealing a formula that would handle any Pt size.

I did stumble upon an great Excel spreadsheet done by a college prof that will let you graphically perform a Least Squares Quadratic Analysis on a data set to determine a formula based on data only. This thing is awesome.

The code would be two layers of case statements. "What font, what Pt size, here is your function." 10 Fonts X 12 Pt size = 120 possibilities plus a default. Ugly code. Ugly code isn't like Ugly Betty or the Ugly Duckling. It never looks better.

The .Net 3.0+ solution with FormattedText looks great, but it can't be considered and if this wasn't fixed now it wasn't going to be.

So, I rearranged my test program to use MeasureString. It spit out a comma separated file with all 10 fonts, 12 pt sizes, and string sizes from 1-79. The string was a 'W' at first, but became an 'M' after I read about em measurements. The graph I really needed was a 3d that showed the pt size and strings size, but I don't know how to make one of these. And then give me a formula on the variables.

I do know how to calculate slope on a line and I can see that Pt size on the same number of characters is a line and not a parabola, sine wave, circle, etc. Dito with the number of characters on the same Pt size.

Slope is m = (y2-y1)/(x2-x1). (Didn't think high school geometry was useful did you?). Slope represents the rate of change. If you know the Pt size and pixel width for 2 characters you can use slope to calculate the pixel width for 3 characters, just by knowing the Pt size. Since these are all lines the slope will be the same from 2 to 3 characters as 2 to 20 characters.

Lets look at Verdana 8pt. The Pt size is fixed and the number of characters are variable. The first pass of the formula works out thusly..

pixelWidth = fieldSize * $m_{\text{for 8pt}}$

Plug in the numbers from the spreadsheet generated by the test program to check the formula.

$22.07639_{\text{for 2 char}}$ = 2 * 9.26042

But, 2 * 9.26042 = 18.52084 and not 22.07639. Perhaps the formula needs some revision. Lets take the difference, 3.55555, and add it as a constant. The formula is now..

pixelWidth = fieldSize * $m_{\text{for 8pt}}$ + 3.55555

Let's jump up to a 10 character width and see if this still holds.

96.15971 = 10 * 9.26042 + 3.55555

96.15971 does not equal 96.15975, but it's close enough. I think we found the formula for calculating the pixel width for Verdana 8pt.

Maybe, the m constant is not quite so constant. If we could get 1 formula for all Pt sizes instead of 18 of them that would really streamline the code.

Let's get the formula for 7 pt. Here's the data for 7 and 8 pt.

Verdana 7 pt

| Num Char | Width | Slope (m) |
|---|---|---|
| 1 | 11.21397 | |
| 2 | 19.31684 | 8.10287 |
| … | | |

| | | |
|---|---|---|
| 10 | 84.13975 | 8.10287 |
| ... | | |

Verdana 8 pt

| Num Char | Width | Slope (m) |
|---|---|---|
| 1 | 11.21397 | |
| 2 | 19.31684 | 9.26042 |
| ... | | |
| 10 | 84.13975 | 9.26042 |
| ... | | |

pixelWidth = fieldSize * $m_{\text{for 7pt}}$ + 3.55555
19.31684 = 2 * 8.10287 + 3.55555

19.31684 does not equal 19.76129. The difference is .44445. A little checking against the other character sizes reveals the 3.55555 is variable on the Pt size of the font. For 7 pt it should be 3.11111. Actually, this constant calculates pretty easily to be pointSize * .44445 and this holds for all the fonts. I suspect that this constant varies on Character Set or Languages.

Lets plug this in and calculate on the 10 character data point.

84.13975 = 10 * 8.10287 + 3.11111

84.13975 is close enough to 84.13981. I think we have the new constants for Verdana 7 pt.

Is there a relationship between the change in the constants from 7 pt to 8 pt? We’ll start with a simple relationship. Calculate the slope of the slopes. This may not be true. The relationship could be a parabola or something else.

(9.26042 – 8.10287)/(8-7) = 1.15755/1 = 1.15755

You can quickly check this against the other point sizes for Verdana and see that 1.1575 is a good $m_{\text{for all points}}$. There’s a faster way to calculate $m_{\text{for all points}}$ than comparing various point sizes on a font. I mean we have to do this 9 other times. There is a shortcut.

$m_{\text{for all pt}} = m_{\text{for pointSize}}/\text{pointSize}$

or to put it another way

$m_{\text{for 8pt}}$ = 8.10287 + 1.15755 = 9.26042

for 8pt

1.15755 = 9.26042/8

So, it checks.

How does this relate to the existing pixelWidth formula?

pixelWidth = fieldSize * $m_{\text{for 8pt}}$ + constant
where $m_{\text{for pointSize}}$ = pointSize * 1.15755
and constant = pointSize * .44445

or the one formuala we have been looking for..

pixelWidth = (fieldSize * pointSize * 1.15755) + (pointSize * .44445)

The formula above holds for all the fonts. The constant, 1.15755, is unique for each font and actually varies by character. 1.15755 is based on ‘M’. ‘W’ will give a slightly higher constant.

The table below holds the constants for all 10 fonts. The formula is always..

pixelWidth = (fieldSize * pointSize * 1.15755) + (pointSize * .44445)

| Font | $m_{\text{for pointSize}}$ |
|---|---|
| arial | 1.14386667 |
| consolas | 0.7552125 |
| courier new | .82421875 |
| georgia | 1.2734375 |
| lucida console | 0.827475 |
| microsoft sans serif | 1.14388 |
| tahoma | 1.0579425 |
| times new roman | 1.2213575 |
| trebuchet | 0.9746 |
| verdana | 1.1575525 |

Here is the test program that generated the data in .Net 2.0 c#. Here is the best of the many, many datasets I generated. It’s a comma separated text file.

Sorry, I can’t list all the blind alleys I walked into. That’s a much longer and probably more useful article. There were 3 days of them.

Good luck with your project. Thanks to my friends for helping with a simple, elegant, one formula solution that fit all the parameters.

This entry was posted on Tuesday, February 19th, 2008 at 9:49 am and is filed under Programming. You can follow any responses to this entry through the RSS 2.0 feed. You can leave a response, or trackback from your own site.

## Leave a Reply

You must be logged in to post a comment.