

Forum:[Hack Forums](#)

Thread: [Spreader | Create Custom SpeedRead Gifs | Gif Encode, Measure String](#)

Odus - August 4, 2015, 6:12 am

After seeing a hilarious use of speed reading as a signature, I was inspired to recreate this in C#. This was a lot of fun to make and I learned some more about the Drawing.Graphics class in the process, which is why I'm sharing it.

So how does this work? We start by splitting up the input by space - to retain punctuation - and building a new frame for each of them. The main issue was font size as some words are larger than others and may spill out of the frame. To fix this, we take the largest word and update the font size to the bitmap's size divided by the width of our measured string (largest word).

For ease, I pinched the BumpKit GifEncoder and used that to encode and export the output.

Ignore the poor front end, it's sloppy and was only for demonstration. Otherwise, I'd like some feedback on the class. I rarely write from scratch in C# so I'm looking forward to what people can pick out.

Demonstration: [\(Click to View\)](#)

[img]http://orig01.deviantart.net/46de/f/2015/216/0/7/sr_demo_by_oduslabs-d9458l8.gif[/img]

Produces:

[img]http://orig11.deviantart.net/c699/f/2015/216/6/7/sr_out_by_oduslabs-d9458pz.gif[/img]

Downloads:

[Full source \(GitHub\)](#)

[Binary only \(MediaFire\)](#)

Credits:

BumpKit GifEncoder:

<https://github.com/DataDink/Bumpkit/blob/master/BumpKit/BumpKit/GifEncoder.cs>

Snoweyes - August 4, 2015, 6:57 am

That's a badass project! In my opinion pretty easy to make but very useful aswell for signatures and stuff.

Can you explain BumpKit? I think i know what it is but i want to be sure.

Odus - August 4, 2015, 7:34 am

(Aug 4, 2015 06:57 AM)Snoweyes Wrote: [\[CITE\]](#) That's a badass project! In my opinion pretty easy to make but very useful aswell for signatures and stuff.

Can you explain BumpKit? I think i know what it is but i want to be sure.

I never said it was a hard project ?

BumpKit is an image manip library with all sorts of goodies to easily do more with the Graphics class. You can read more on their Git readme: <https://github.com/DataDink/Bumpkit>

But rather than carry an entire lib, I just took the GifEncoder class.

b?xxy - August 4, 2015, 8:11 am

Nice project you're working on, thanks for the PM.

Corwin - August 4, 2015, 11:26 am

This is a pretty nice program, I love it already.

Serux - August 4, 2015, 12:05 pm

A few things.

1: Where the fuck are the namespaces?

2: LINQ is pretty cool, try using it.

Your BuildFrames return is

Code foreach (var word in words)

yield return RenderFrame(word);Becomes:

Code return words.Select(RenderFrame);3: Add some error handling. I see a lot of unhandled exceptions in the making.

I was going to harass you about passing in a stream as a constructor but you are implementing IDisposable and correctly using a using statement so thats a nice surprise.

There's some quality of life changes such as incorporating a few design patterns to make your code more accessible.

Not bad overall though. Much higher quality than what I was expecting. Main thing is just to handle your exceptions and you're good.

SIM-1676753 - August 4, 2015, 12:07 pm

(Aug 4, 2015 11:50 AM)Odus Wrote: [\[CITE\]](#) Oops that slipped by me, fixed.

Good point, done.

Also done, I just read into those. Very cool. Thank you Trvp :)

Also, since you are calling RenderFrame in a for each loop, I would pass the brush as a parameter so you don't initialize the same brush multiple times.

Zer0Fact0r - August 4, 2015, 12:41 pm

Some feedback and nitpicks for ReaderEngine:

(Aug 4, 2015 12:07 PM)Mr. Trvp Wrote: [\[CITE\]](#) Also, since you are calling RenderFrame in a for each loop, I would pass the brush as a parameter so you don't initialize the same brush multiple times.

Instead you should have the brush be a private instance variable and have the ForeColor property interface with Brush.Color.

<https://github.com/oduslabs/Speed-Reader/blob/master/SpeedReader/Classes/ReaderEngine.cs#L19>

I think you meant to name fontCol foreCol. Also it seems kind of weird that you would let the class be constructed with a ForeColor and BackColor but not with Font and Size seeing as how they're all public properties and all the constructor parameters are optional anyway.

Also it's generally better to be verbose (at least, in C#) or at least consistent in your naming style. fontCol should be foreColor, backCol should be backColor (plus this makes it consistent with the property names), maxWrd should be maxWord and so forth. And compare line 64-74 with lines

45-58: you chose to shorten graphic to g, brush to b, fileStream to fs, yet you spelled out stream and encoder.

<https://github.com/oduslabs/Speed-Reader/blob/master/SpeedReader/Classes/ReaderEngine.cs#L45>

Why write the gif to a MemoryStream then to a FileStream when you can write directly to the FileStream?

Serux - August 4, 2015, 2:28 pm

So question. Are you just abusing the nullable variables or is there actually a reason you're doing something like this

```
Code public ReaderEngine(Color? fontCol = null, Color? backCol = null)
{
    Font = new Font("Arial", 120, FontStyle.Regular);

    ForeColor = fontCol ?? Color.Black;
    BackColor = backCol ?? Color.White;
}
```

Why not use have two constructors - one that takes no parameters and sets forecolor to black and so on and one that takes two variables and sets them to the colors.

Also going off what zer0 was saying is that you should really never have variables from initialized parameters and then be able to set variables later on down the road. Pick one. Either let the user change the variables on the fly and set them to be public or private readonly them all and initialize from constructor.

I shouldn't really be able to do something like this:

```
Code var reader = new ReaderEngine(Color.Black, Color.White)
{
    Font = "Arial",
    ForeColor = Color.Purple,
    BackColor = Color.White
}
```

I think this would be a good case where you could use the factory pattern efficiently.

Odus - August 4, 2015, 4:02 pm

(Aug 4, 2015 12:05 PM)Serux Wrote: [\[CITE\]](#) 2: LINQ is pretty cool, try using it.

Your BuildFrames return is

Code foreach (var word in words)

yield return RenderFrame(word);Becomes:

Code return words.Select(RenderFrame);

You know what, I remember saying exactly the same thing with something like this, no idea why I forgot. Very clever, thanks. LINQ was already present though, see maxWord In

(Aug 4, 2015 12:05 PM)Serux Wrote: [\[CITE\]](#) There's some quality of life changes such as incorporating a few design patterns to make your code more accessible.

Not bad overall though. Much higher quality than what I was expecting. Main thing is just to handle your exceptions and you're good.

I agree, will work on that later.

(Aug 4, 2015 12:41 PM)Zer0Fact0r Wrote: [\[CITE\]](#) Instead you should have the brush be a private instance variable and have the ForeColor property interface with Brush.Color.
Very good idea, implemented it.

(Aug 4, 2015 12:41 PM)Zer0Fact0r Wrote: [\[CITE\]](#) I think you meant to name fontCol foreCol. Also it seems kind of weird that you would let the class be constructed with a ForeColor and BackColor but not with Font and Size seeing as how they're all public properties and all the constructor parameters are optional anyway.

These were named like such since the names I'd like to use were already in use. Taking Trvp's advice of auto properties freed them up and I forgot to change them. As for font -> fore, the property was originally named 'FontColor' but I changed it last minute since it never really sat with me. Mismatching conventions have also been fixed.

(Aug 4, 2015 12:41 PM)Zer0Fact0r Wrote: [\[CITE\]](#) Why write the gif to a MemoryStream then to a FileStream when you can write directly to the FileStream?

Fantastic idea. I forgot they both derive from Stream.

(Aug 4, 2015 02:28 PM)Serux Wrote: [\[CITE\]](#) So question. Are you just abusing the nullable variables or is there actually a reason you're doing something like this

It was a workaround since the compiler wasn't happy my optional args weren't 'compile-time constants'. Got rid of all optional and just initialized instead. As for error handling, inadvertently got most of it covered since all my properties & private variables are initialized. I've also added some checks for null arguments.

Thanks very much to both of you for your feedback. I'm going to post more in C#, never had this kind of criticism before. Fixes have been pushed to master.

Ethan - August 4, 2015, 6:40 pm

This is fun to play around with, good work Odus!

Sleeping With Sirens - August 5, 2015, 12:14 am

"I sexually identity as an Attack helicopter" Imfao.

Regardless, nice release.