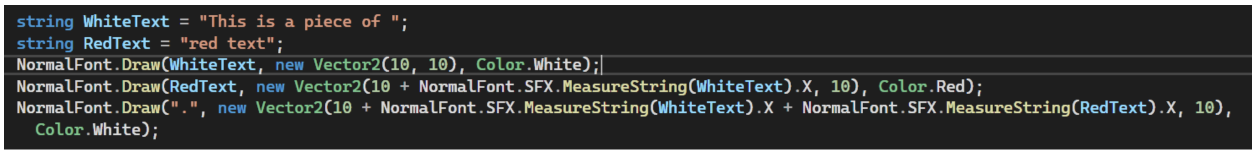
Introduction

This class is rather massive, so it requires an individual page for it to document.

Inspired by Scribble, this class is built specifically for text rendering in the UF-Ex engine. Most of you should know by now that the text drawing/typing functions/classes are less than ideal to use, including a multitude of issues:

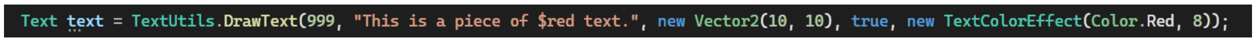
- Lack of rendering options
- Lack of variety (You can use LimitDraw, you can also use CentreDraw, but you cannot combine both)
- Lack of readability (Whether it is due to the lack of documentation or comments in the code itself, or just the string command itself)
- Lack of function

Of course, you can use the existing functions to draw some simple texts, like
“This is a piece of text.”
However, if you would like to draw texts like
“This is a piece of red text ”
You would either do one of the following



```
string WhiteText = "This is a piece of ";
string RedText = "red text";
NormalFont.Draw(WhiteText, new Vector2(10, 10), Color.White);
NormalFont.Draw(RedText, new Vector2(10 + NormalFont.SFX.MeasureString(WhiteText).X, 10), Color.Red);
NormalFont.Draw(".", new Vector2(10 + NormalFont.SFX.MeasureString(WhiteText).X + NormalFont.SFX.MeasureString(RedText).X, 10),
  Color.White);
```

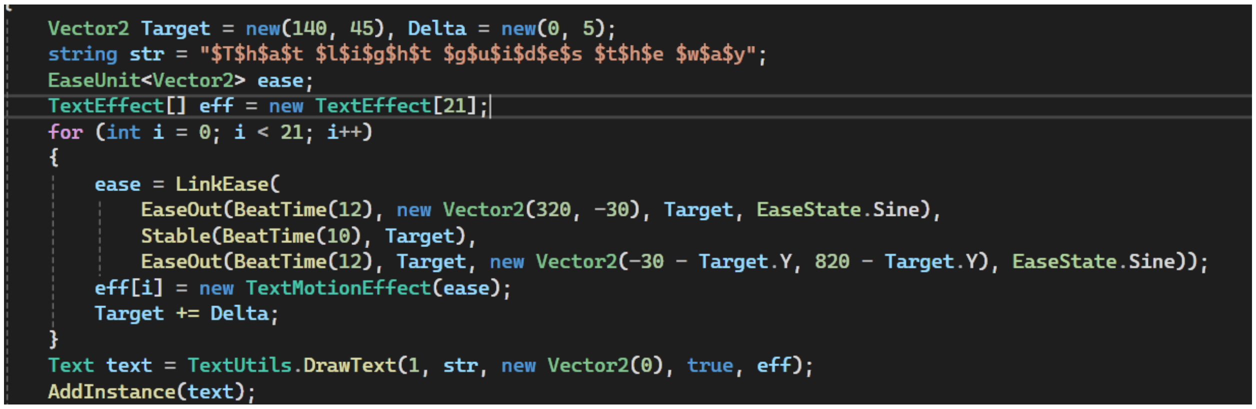
Or



```
Text text = TextUtils.DrawText(999, "This is a piece of $red text.", new Vector2(10, 10), true, new TextColorEffect(Color.Red, 8));
```

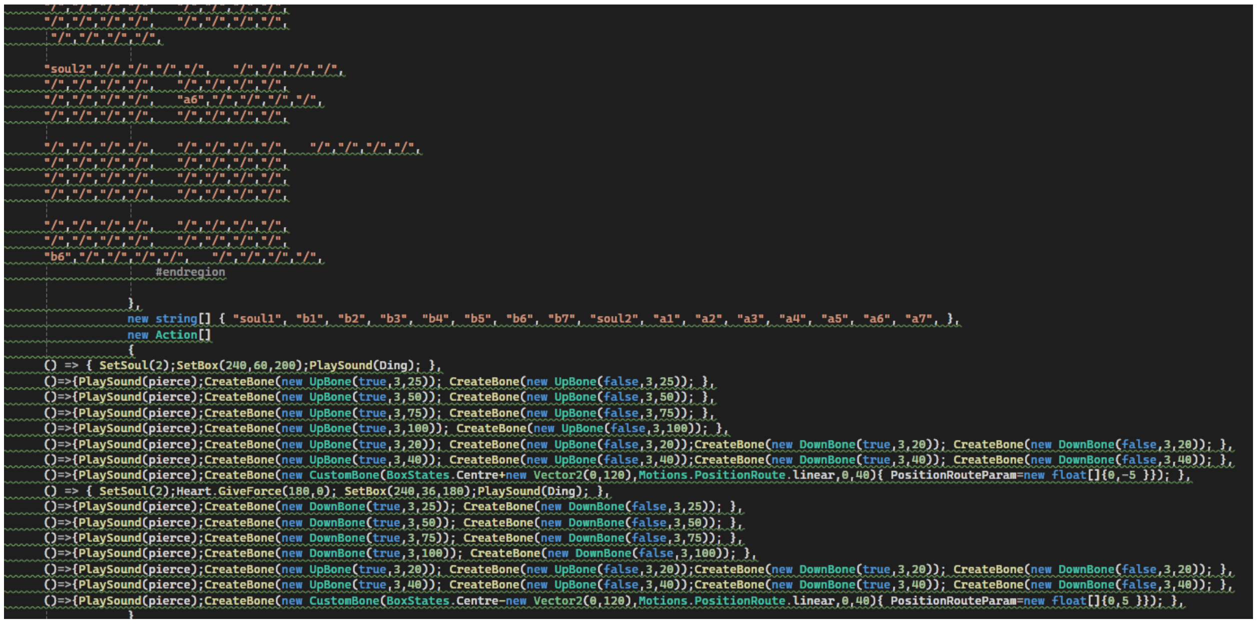
The solution displayed in the first image is clearly not ideal, if you think that this is fine, you can image if you wanted to display a text that looks like
”Green Black Red Yellow”
And suddenly the amount of code required is tripled, and it will become more likely to lag.

The solution in the second image may look better, however if you want to create a text effect that looks like this (Credit to Sedwix), the code would look like this.



```
    Vector2 Target = new(140, 45), Delta = new(0, 5);
    string str = "$T$h$a$t $l$i$g$h$t $g$u$i$d$e$s $t$h$e $w$a$y";
    EaseUnit<Vector2> ease;
    TextEffect[] eff = new TextEffect[21];
    for (int i = 0; i < 21; i++)
    {
        ease = LinkEase(
            EaseOut(BeatTime(12), new Vector2(320, -30), Target, EaseState.Sine),
            Stable(BeatTime(10), Target),
            EaseOut(BeatTime(12), Target, new Vector2(-30 - Target.Y, 820 - Target.Y), EaseState.Sine));
        eff[i] = new TextMotionEffect(ease);
        Target += Delta;
    }
    Text text = TextUtils.DrawText(1, str, new Vector2(0), true, eff);
    AddInstance(text);
```

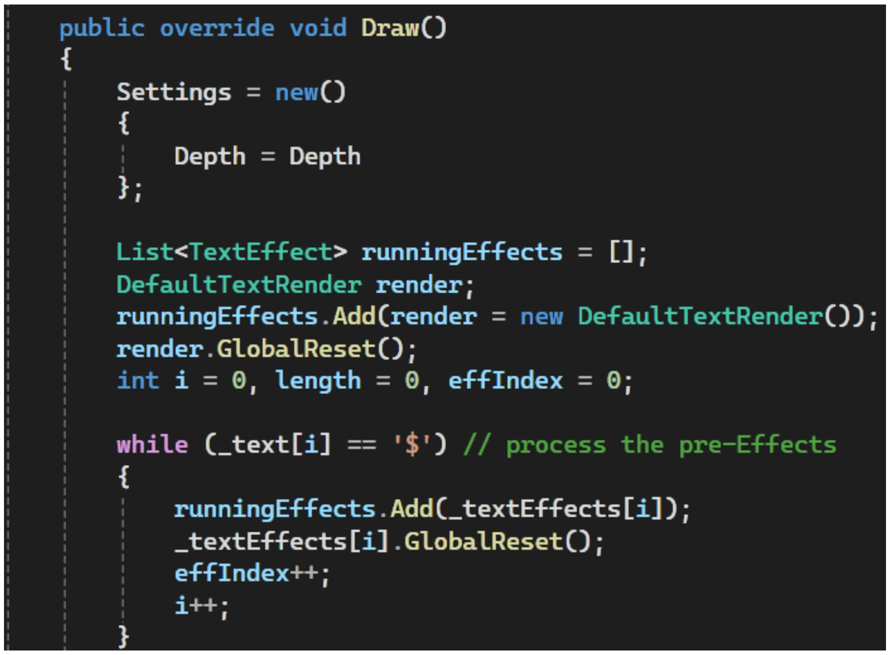
This is clearly not desirable as it can be easily simplified if there was a function that would be executed per character in the text.
Furthermore, each “$” can easily mean different things and may not be immediately readable before looking at the array of arguments, which will become increasingly difficult to search for the desired index due to its length. (Similar issue is present with Extends.RhythmCreate and Extends.SpecialRhythmCreate)



```
    "/","/","/","/",   "/","/","/","/",
     "/","/","/","/",

    "soul2","/","/","/","/",   "/","/","/","/",
    "/","/","/","/",   "/","/","/","/",
    "/","/","/","/",   "a6","/","/","/","/",
    "/","/","/","/",   "/","/","/","/",

    "/","/","/","/",   "/","/","/","/",   "/","/","/","/",
    "/","/","/","/",   "/","/","/","/",
    "/","/","/","/",   "/","/","/","/",
    "/","/","/","/",   "/","/","/","/",

    "/","/","/","/",   "/","/","/","/",
    "/","/","/","/",   "/","/","/","/",
    "b6","/","/","/","/",   "/","/","/","/",
                #endregion

            },
            new string[] { "soul1", "b1", "b2", "b3", "b4", "b5", "b6", "b7", "soul2", "a1", "a2", "a3", "a4", "a5", "a6", "a7", },
            new Action[]
            {
    () => { SetSoul(2);SetBox(240,60,200);PlaySound(Ding); },
    ()=>{PlaySound(pierce);CreateBone(new UpBone(true,3,25)); CreateBone(new UpBone(false,3,25)); },
    ()=>{PlaySound(pierce);CreateBone(new UpBone(true,3,50)); CreateBone(new UpBone(false,3,50)); },
    ()=>{PlaySound(pierce);CreateBone(new UpBone(true,3,75)); CreateBone(new UpBone(false,3,75)); },
    ()=>{PlaySound(pierce);CreateBone(new UpBone(true,3,100)); CreateBone(new UpBone(false,3,100)); },
    ()=>{PlaySound(pierce);CreateBone(new UpBone(true,3,20)); CreateBone(new UpBone(false,3,20));CreateBone(new DownBone(true,3,20)); CreateBone(new DownBone(false,3,20)); },
    ()=>{PlaySound(pierce);CreateBone(new UpBone(true,3,40)); CreateBone(new UpBone(false,3,40));CreateBone(new DownBone(true,3,40)); CreateBone(new DownBone(false,3,40)); },
    ()=>{PlaySound(pierce);CreateBone(new CustomBone(BoxStates.Centre+new Vector2(0,120),Motions.PositionRoute.linear,0,40){ PositionRouteParam=new float[]{0,-5 }}); },
    () => { SetSoul(2);Heart.GiveForce(180,0); SetBox(240,36,180);PlaySound(Ding); },
    ()=>{PlaySound(pierce);CreateBone(new DownBone(true,3,25)); CreateBone(new DownBone(false,3,25)); },
    ()=>{PlaySound(pierce);CreateBone(new DownBone(true,3,50)); CreateBone(new DownBone(false,3,50)); },
    ()=>{PlaySound(pierce);CreateBone(new DownBone(true,3,75)); CreateBone(new DownBone(false,3,75)); },
    ()=>{PlaySound(pierce);CreateBone(new DownBone(true,3,100)); CreateBone(new DownBone(false,3,100)); },
    ()=>{PlaySound(pierce);CreateBone(new UpBone(true,3,20)); CreateBone(new UpBone(false,3,20));CreateBone(new DownBone(true,3,20)); CreateBone(new DownBone(false,3,20)); },
    ()=>{PlaySound(pierce);CreateBone(new UpBone(true,3,40)); CreateBone(new UpBone(false,3,40));CreateBone(new DownBone(true,3,40)); CreateBone(new DownBone(false,3,40)); },
    ()=>{PlaySound(pierce);CreateBone(new CustomBone(BoxStates.Centre-new Vector2(0,120),Motions.PositionRoute.linear,0,40){ PositionRouteParam=new float[]{0,5 }}); },
            }
```

Code from Let’s Go Now (Tlott)

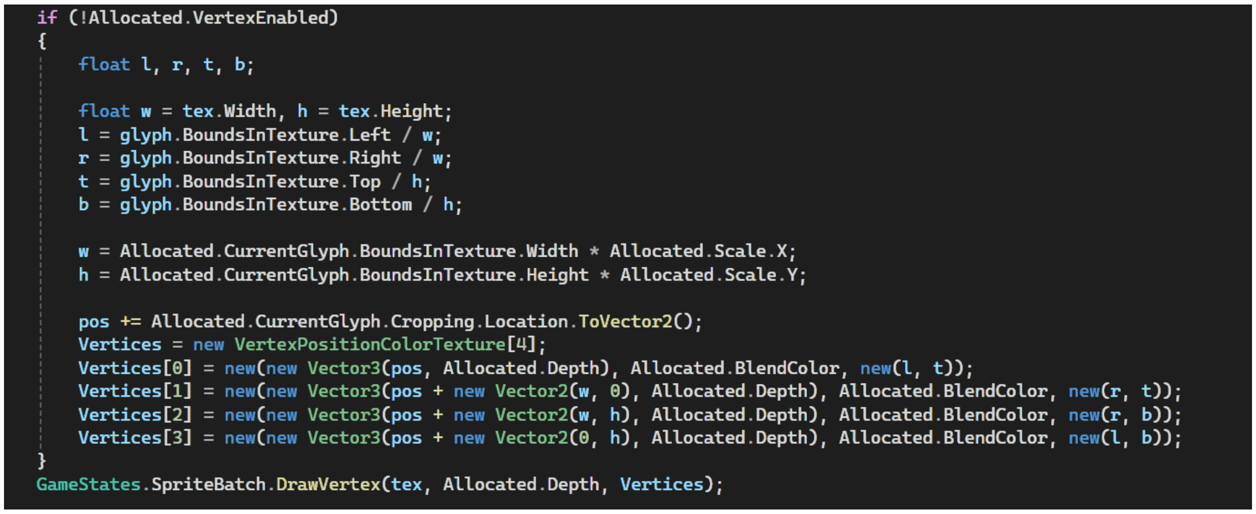
So, why is this any better than the existing TextUtils?
The rendering process in TextUtils is unoptimized



```
public override void Draw()
{
    Settings = new()
    {
        Depth = Depth
    };

    List<TextEffect> runningEffects = [];
    DefaultTextRender render;
    runningEffects.Add(render = new DefaultTextRender());
    render.GlobalReset();
    int i = 0, length = 0, effIndex = 0;

    while (_text[i] == '$') // process the pre-Effects
    {
        runningEffects.Add(_textEffects[i]);
        _textEffects[i].GlobalReset();
        effIndex++;
        i++;
    }
```

As you can see, the text effects are processed per frame. This is clearly poor coding, as the text effects are static and can be parsed during initialization, instead of parsing them per frame.

The code also does not make sense



```
if (!Allocated.VertexEnabled)
{
    float l, r, t, b;

    float w = tex.Width, h = tex.Height;
    l = glyph.BoundsInTexture.Left / w;
    r = glyph.BoundsInTexture.Right / w;
    t = glyph.BoundsInTexture.Top / h;
    b = glyph.BoundsInTexture.Bottom / h;

    w = Allocated.CurrentGlyph.BoundsInTexture.Width * Allocated.Scale.X;
    h = Allocated.CurrentGlyph.BoundsInTexture.Height * Allocated.Scale.Y;

    pos += Allocated.CurrentGlyph.Cropping.Location.ToVector2();
    Vertices = new VertexPositionColorTexture[4];
    Vertices[0] = new(new Vector3(pos, Allocated.Depth), Allocated.BlendColor, new(l, t));
    Vertices[1] = new(new Vector3(pos + new Vector2(w, 0), Allocated.Depth), Allocated.BlendColor, new(r, t));
    Vertices[2] = new(new Vector3(pos + new Vector2(w, h), Allocated.Depth), Allocated.BlendColor, new(r, b));
    Vertices[3] = new(new Vector3(pos + new Vector2(0, h), Allocated.Depth), Allocated.BlendColor, new(l, b));
}
GameStates.SpriteBatch.DrawVertex(tex, Allocated.Depth, Vertices);
```

If `VertexEnabled` is false, vertices will be used.
Uh oh.

There is not much to talk about since there is not much code to begin with, however this also raises another issue, the lack of features. You can only draw text and apply some basic effects onto them.