# Rendering XAML to a JPEG using Silverlight 3

Something that had been relatively easily do in WPF that was near impossible in silverlight was to easily take the XAML that had been rendered on the screen and allow the user to save it as an image file.

With the release of version 3 of silverlight this is now possible through the use of the WriteableBitmap class. Once the XAML has been rendered to the to the writablebitmap it can then be written to a Bitmap file or using a third party code encoded as a JPEG or PNG as shown below. This can have a range of applications form allowing users create avitars through to saving an image of a silverlight game. Once you have the image it can then be saved to isolated storage, a web server via a web service or to the users own machine.


In the sample I have used the FJcore library to encode it as a JPEG to keep the file small, however it is just as easy to use the Joe Stegman's PNG encoder.

The first step is to write your XAML to the writeable bitmap which can easily be done in one line through the constructor. To render the image identical to the screen we pass null into the transform argument.

```
WriteableBitmap bitmap = new WriteableBitmap(sourceElement, null);
```

Using a type of panel such as a canvas or grid will allow you can render multiple elements together in one image. One thing to note is that it ignores the Background property of the pannel placing all elements on default bitmap black background. To get around this you can first drawn a white rectangle as a background to your image.

The easiest way to save the image and get the best mix of file size and quality is to use the FjCore Library which is design to be a simple lightweight JPEG encoder/decoder for use with silverlight and distributed under the MIT License licence.

The fjcore encode method expect a byte array of the image so the only additional method in the project converts the WriteableBitmap (using a variation on code provided by RHLopez in a stack overflow question) and adds it to a file stream which in the case of this sample is created from the save dialog class so is inturn saved to the users machine.

```
private static void SaveToFile(WriteableBitmap bitmap,Stream fs)
{
            //Convert the Image to pass into FJCore
            int width = bitmap.PixelWidth;
            int height = bitmap.PixelHeight;
            int bands = 3;

            byte[][,] raster = new byte[bands][,];

            for (int i = 0; i &amp;lt; bands; i++)
            {
                raster[i] = new byte[width, height];
            }

            for (int row = 0; row &amp;lt; height; row++)
            {
                for (int column = 0; column &amp;lt; width; column++)
                {
                    int pixel = bitmap.Pixels[width * row + column];
                    raster[0][column, row] = (byte)(pixel &amp;gt;&amp;gt; 16);
                    raster[1][column, row] = (byte)(pixel &amp;gt;&amp;gt; 8);
                    raster[2][column, row] = (byte)pixel;
                }
            }

            ColorModel model = new ColorModel {colorspace = ColorSpace.RGB };

            FluxJpeg.Core.Image img = new FluxJpeg.Core.Image(model, raster);

            //Encode the Image as a JPEG
            MemoryStream stream = new MemoryStream();
            FluxJpeg.Core.Encoder.JpegEncoder encoder = new FluxJpeg.Core.Encoder.JpegEnc

            encoder.Encode();

            //Move back to the start of the stream
            stream.Seek(0, SeekOrigin.Begin);

            //Get the Bytes and write them to the stream
            byte[] binaryData = new Byte[stream.Length];
            long bytesRead = stream.Read(binaryData, 0, (int)stream.Length);

            fs.Write(binaryData, 0, binaryData.Length );

        }
```

The full code sample can be downloaded here