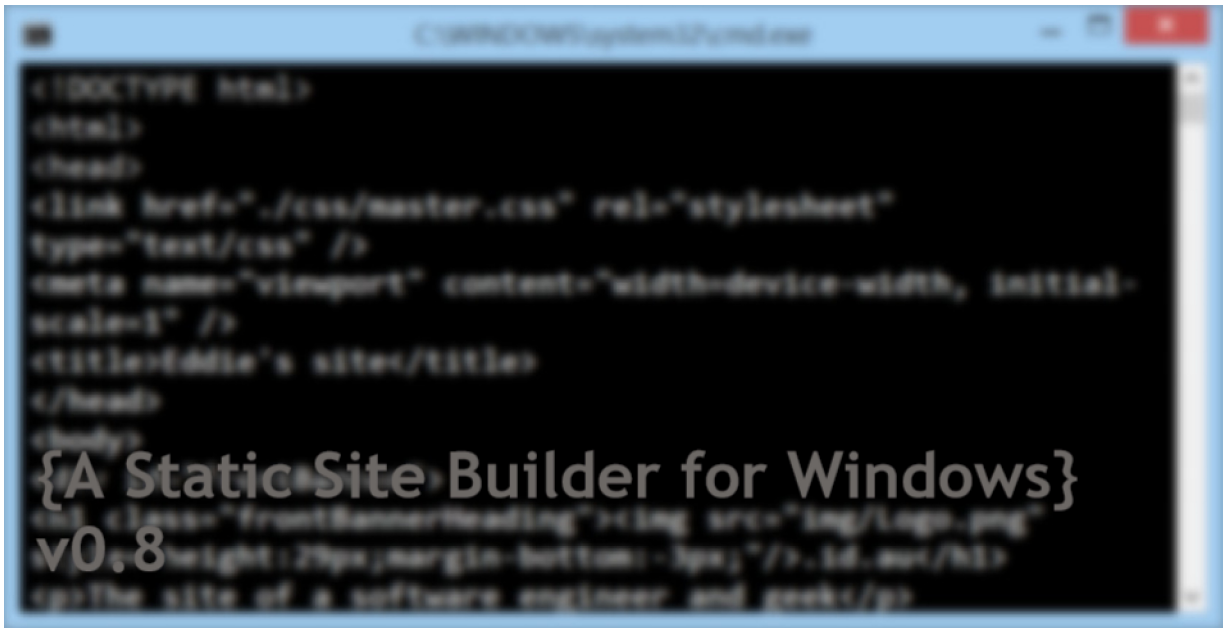

Published 2015/05/18 in Software

Today I'm writing about the tool that I use to build this site, which fills the role of a static CMS. It lets me write in the friendly Markdown notation to create articles quickly, throw the applicable images in a directory and run a script to rebuild the site content.

How it works

Like other static CMS systems, this tool (yet to be properly named) uses text based content and templates which are combined to produce a (hopefully) pretty site. Everything is organised into directories.

Paths file

Everything starts with a Paths.txt file, that looks a lot like the one below:

```
#This file specifies paths to the site content and output directories
articlesDir=C:\Website\Content\
macrosDir=C:\Website\Macros\
templatesDir=C:\Website\Templates\
virtualArticlesDir=C:\Website\Virtual Content\
outputDir=C:\WebsiteOutput\
```

This file is expected to be in the same directory as the site builder executable, and specifies where all of the data is or will go.

Content

The content directory contains a collection of text files with the .md file extension, expected to be in an enhanced Markdown notation. Each file should also have a preamble that borrows a subset of YAML, namely associative arrays.

An article may look like this:

```
---
Date:2015-05-05
Category:Software
```

```
Title:My awesome project
Template:Post
---
Today I wrote an awesome app that does...
```

The markdown processing is done by Jeff Atwood's extremely useful MarkdownSharp library, which is apparently a descendent of Milan Negovan's Markdown .NET.

Macros

Macros are files that live in the directory specified by macrosDir, and can be referred to by articles or templates to add common or variable content. Articles (and other content) may contain references to Markdown macros, the syntax of which I borrowed from Aaron Parecki's post Some Enhancements to Markdown. The syntax to use a macro is `![:macroName parameter1 parameter2](value)`. The only mandatory element is the macro name. If a parameter would need to contain spaces, these can be substituted for underscores. If a parameter would need to contain underscores, these can be substituted for double-underscores.

For example if one wanted to include a reference to a YouTube video, there's a macro to do that, and it would be included in the article like so:

```
![:youtube 1366 768](https://www.youtube.com/embed/M1ufW2INWmM)
```

This would create a YouTube embed pointing to the desired URL. There are two main types of macro available:

Text macros

This category contains both plaintext (.txt) and Markdown (.md) macros. Both work in the same way, but the Markdown macro passes its output through MarkdownSharp before returning the result. The principle method these macros use is string replacement. For example the YouTube embed macro is simply:

```
<iframe width="%p1%" height="%p2%" src="%v1%"
frameborder="0"></iframe>
```

Where %p#% stands for a parameter, and %v#% stands for a (the) value. Macros may also incorporate any data from the article properties (including those inherited from the template). If the article had a "Title" attribute, the %Title% would work too. In all cases the named value is replaced with the real value - simple. If the macro refers to something that doesn't exist, then it is not replaced. So text macros are really simple to set up, but they are limited in that they only get access to the current article's information. XSLT macros are the opposite.

XSLT Macros

XSLT macros are more complicated to write but get access to the metadata for *every* article. This makes them useful for features such as site navigation. The file extension for a XSLT macro is .xsl, and the file is expected to be a standard XML transform using XPath 1.0 expressions. This is built on .NET's included XSL engine. XSLT macros work by running a transformation over a dynamically constructed XML document, which at its simplest looks like the example below:

```
<Articles>
    <Article>
        <!--Every attribute of the article is listed in an XML node
here-->
        <Title>My article</Title>
        <Date>2015-02-05</Date>
    </Article>
    <Article>
        <Title>My follow-up article</Title>
        <Date>2015-02-10</Date>
    </Article>
<Articles>
```

It is possible to group and sort articles. Another text file with the same name as the macro's XSL file, but with a .groups extension, may be included. If found, this will cause the XML to be constructed differently.

For example, a .groups file containing:

```
Date Descending
```

would cause the following XML to be generated for the macro:

```
<Articles>
    <Group Attribute="Date" Value="2015-02-05">
        <Article>
            <Title>My article</Title>
            <Date>2015-02-05</Date>
        </Article>
    </Group>
    <Group Attribute="Date" Value="2015-02-10">
        <Article>
            <Title>My follow-up article</Title>
            <Date>2015-02-10</Date>
        </Article>
    </Group>
<Articles>
```

Any number of levels of grouping may be specified, in which case groups are nested inside of each other.

If there is a need to combine features of both macro types, it is possible for a macro's output to include a macro invocation in the same format as that used for an article.

Templates

Where an article includes a Template attribute, the tool will try to find a template with the same name in the templates directory. It is not mandatory for an article to use a template, but it makes life much easier. At this stage, only Markdown templates (.mdt) are supported.

A template may look like this:

```
---
Tags:BlogPost
---
<!DOCTYPE html>
<html>
<head>
<title>![:articleTitle]</title>
</head>
<body>
<div id="divMain" class="main">
%content%
</div>
</body>
</html>
```

The %content% marker is used to determine where the article content should be placed after processing. It is a moot point at present, but the article content does not have use the same processing engine as the template.

Properties specified in the template are made available to macros running in the associated article, and vice-versa.

Virtual Articles

Virtual articles are a way to generate pages based on the metadata of all loaded articles. They are based on XSL as well, and are expected to generate a namespace-less XML document containing an Articles tag, which in turns contains Article tags, containing Content tags. Within each Content tag there is expected to be a Markdown article in the same format as a regular article. The file extension for a virtual article generator stylesheet is .xslmd .

An example article generator stylesheet to produce category article lists is:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-result-
prefixes="msxsl"
>
  <xsl:output method="xml" indent="no"/>
  <!--Note: Indentation of article content is important here. Any
indentation preceding the article content (after the preamble) will
cause the Markdown
  engine to treat HTML markup as code (and in turn escape it for
display on the web) rather than as HTML.-->

  <xsl:template match="/">
    <xsl:element name="Articles">
      <xsl:for-each select="//Group[@Value != '']">
        <xsl:element name="Article">
          <xsl:element name="Content">
---
Title:Posts in <xsl:value-of select="@Value"/>
Template:categoryPage
Path:Category-<xsl:value-of select="@Value"/>.html
---
```

```
\![:categoryPageThumbnails <xsl:value-of select="translate(@Value, '
', '_')"/>]
          </xsl:element>
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

This stylesheet has an associated .groups file, containing just one entry:

```
Category
```

The input XML for this stylesheet would look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<Articles>
  <Group Attribute="Category" Value="">
    <Article>
      <Title>Eddie's Site</Title>
      <Category />
      <Tags />
      <Date />
      <DateModified />
      <HeadingImage />
      <Template>index</Template>
      <Path />
    </Article>
  </Group>
  <Group Attribute="Category" Value="Electronics">
    <Article>
      <Title>Inside the ASUS Taichi Power Plug</Title>
      <Category>Electronics</Category>
      <Tags>
        <Value>Electronics</Value>
        <Value> Computing</Value>
      </Tags>
      <HeadingImage>img/TaichiHeader</HeadingImage>
      <Template>Post</Template>
      <Path />
    </Article>
    <Article>
      <Title>Xbox Controller Repair</Title>
      <Category>Electronics</Category>
      <Tags>
        <Value>Gaming</Value>
        <Value>Electronics</Value>
      </Tags>
      <HeadingImage>img/XbCR1</HeadingImage>
      <Template>Post</Template>
      <Path />
    </Article>
  </Group>
  <Group Attribute="Category" Value="Site News">
    <Article>
```

```
      <Title>Goodbye Old Site</Title>
      <Category>Site News</Category>
      <Tags>
        <Value>Site News</Value>
      </Tags>
      <HeadingImage>img/OldEddiesoftSite-banner</HeadingImage>
      <Template>Post</Template>
      <Path />
    </Article>
  </Group>
</Articles>
```

For which the output would be:

```
<Articles><Article><Content>
---
Title:Posts in Electronics
Template:categoryPage
Path:Category-Electronics.html
---
![:categoryPageThumbnails Electronics]
          </Content></Article><Article><Content>
---
Title:Posts in Site News
Template:categoryPage
Path:Category-Site News.html
---
![:categoryPageThumbnails Site_News]
          </Content></Article></Articles>
```

The article generator then extracts every Article node and processes its content through the Markdown article processor. The actual content generation is left to a macro named categoryPageThumbnails which has an associated groups file.

Image Processing

The tool is able to resize images in a given directory for display on different devices. That means high-resolution photos can be just be dropped in the site content and the site builder can take care of ensuring that appropriately sized images are available for different devices.

Image processing is driven by another two other configuration files that are expected to be in the same directory as the executable, named ImagePaths.txt and ImageSizes.txt.

ImagePaths.txt

This file is quite simple, is simply indicates where the input and output directories are:

```
#Format is [source|destination]=<path>
source=C:\Website\Content\img\
destination=C:\WebsiteOutput\img\
```

ImageSizes.txt

This file indicates which sizes the each image in the processing directory should be sized to. Normally each file is output to various different sizes, with a different extension.

```
#Format is <width>x<height> <suffix>
1920x1080
800x600 -sml 75%
300x240:crop -tmb 75%
```

The above file requests the following: - Any file larger than 1920x1080 should be scaled to fit and saved on the original file name. For JPEGs, the default save quality is 95%. - And file larger than 800x600 should be scaled to fit and saved with "-sml" added to the end of the file name before the extension. If JPEG save at 75% quality. - Any file larger than 300x240 will be scaled and then cropped to exactly fit 300x240, and saved with "-tmb" added before the file extension. IF JPEG, save at 75% quality.

At this stage the following file extensions are supported:

- JPEG
- JPG
- PNG
- BMP
- GIF

Images are always saved in their original format.

Processing Sequence

1. Load article content, templates and virtual article generators.
2. Create "virtual" articles and add to main article list.
3. Apply templates. Also copies properties from template to article.
4. Assign output path to "Path" property on each article.
5. Run macros (up to 20 iterations).
6. Generate final HTML.
7. Write output to files.

Limitations

Current limitations (to be improved in future) are: - Only flat file structures are supported. That is the tool doesn't look in nested directories, nor can content be organised into nested directories during output.

Download

For the moment, the tool is available from here. Soon I'm planning to put the source up on GitHub for others to use and improve as they see fit.

Why

...a static site?

Static websites have their advantages, fer particular types of content. A static site can be deployed virtually anywhere (even on ultra-basic hosting services), and can be served extremely fast because there is no server-side processing to do. For a site that is largely a one-way dialogue like this one, it works well.

...build a new tool?

Although Jekyll and similar tools are a good fit in theory, they generally assumed the user was on a Mac, with Ruby, and thus were super-easy to set up on that platform. They could be made to work on Windows, but I wanted a one-step solution that could work out of the box on Windows and didn't depend on any particular web stack being present.

...isn't there an editor built in?

For three reasons:

- Markdown is intended to be readily human readable.
- Because good Markdown-aware text editors already exist. I use MarkdownPad 2.
- I want to keep this project as simple as possible which in turn means that it shouldn't include features that would duplicate the functionality of other tools.

... doesn't it do X?

Because I wrote it to suit my needs, and my needs don't yet cover X. I did design it to be fairly flexible and extensible though, so once the code is online feel free to add X.