Glyph Handbook

by *Fabio Cevasco*

February 2010

Table of Contents

Introduction

Glyph is a *Rapid Document Authoring Framework*.

Think of it like a sort of Ruby on Rails but for creating text documents instead of web sites. With Glyph, you can manage your documents tidily in *projects* that can be used to generate files in a variety of output formats, such as HTML or PDF.

Rationale

Glyph was created to provide an easy way to produce printable documents in HTML and PDF format, using technologies familiar to web developers like HTML and CSS, without using them explicitly. In this very specific context, you could consider Glyph as a possible alternative to LaTeX, bearing in mind that it offers far less features and it is not a typesetting system.

At the time of writing, the most common ways to create a printable document like, say, a PDF file, are the following:

- Use a word processor like Microsoft Word convert it to PDF, for example using Adobe Acrobat — Perhaps the most user-friendly method, but relying on proprietary technologies and GUIs.
- Use LaTeX — It can be used to produce amazing documents, but it is not exactly the most user-friendly option: you have to learn LaTeX first, and you may not want to.
- Use HTML and a 3rd party renderer like PrinceXML.

The third option is arguably the best, because:

- It allows web developers to use what they know best (HTML and CSS).
- It does not rely necessarily on GUIs, thereby giving the author more control on the document formatting and its structure.

The main problem with this method is that writing HTML code can be tedious, but luckily there are plenty of lightweight markup languages like Textile or Markdown that can make writing HTML a fun task. Unfortunately though, a markup language alone is not enough, because:

- It is not a complete alternative to HTML: you have to fallback to HTML to do certain things like the document *head* and *body* tags, and often block-level elements like *div* or *table* are not supported at all.
- They cannot be easily extended to provide additional functionalities, like creating a TOC automatically, for example.

That's where Glyph comes in: Glyph uses a simple but effective macro language to handle those tasks that are not commonly managed by lightweight markup languages, such as:

- Creating TOC, index and bibliography automatically
- Defining the document structure
- Managing text snippets, notes, ...
- Validating links

Chapter I – Basic Usage

This chapter explains how to install and use Glyph to create and manage documents.

1.1 Installing Glyph

To install Glyph, simply run `gem install glyph`, like with any other Ruby gem. Then, create a new directory and initialize a new Glyph project, like so:

`mkdir` *my_glyph_project*

`cd` *my_glyph_project*

```
glyph init
```

That's it. The `my_glyph_project` directory contains your brand new Glyph project. If you open the `document.glyph` file with your favorite text editor, you can start writing your book or article right away.

Glyph's dependencies

Glyph requires the following gems:

- extlib
- gli
- treetop
- rake

Additionally, some Glyph macros may require additional gems, such as:

- RedCloth (*textile* macro)
- Maruku *or* Kramdown *or* BlueCloth (*markdown* macro)
- Haml (if you want to load .sass files with the *style* macro)

1.2 Introducing document.glyph

Every Glyph project contains a `document.glyph` file that is typically used to define the document structure. The default `document.glyph`
generated automatically when creating a new project is the following:

```
document[
  head[style[default.css]]
```

```
  body[
    titlepage[
      title[]
      author[]
      pubdate[]
    ]
    frontmatter[
      toc[]
      preface[header[Preface]
        @[preface.textile]
      ]
    ]
    bodymatter[
      chapter[header[Chapter #1]
        @[chapter_1.textile]
      ]
      chapter[header[Chapter #2]
        @[chapter_2.textile]
      ]
    ]
    backmatter[
      appendix[header[Appendix A]
        @[appendix_a.textile]
      ]
    ]
  ]
]
```

Even without knowing anything about Glyph Language, you can easily figure out that this file defines a document with a Table of Contents, a Preface and some Chapters. `frontmatter[]`, `preface[]`, `chapter[]`, etc. are all Glyph *macros* used to define — in this case — some structural elements. In practice, this means that if you plan to generate an HTML document, they'll be converted into `<div>` tags.

Be aware that other macros, on the other hand, are used to do something completely different, e.g.:

- `toc[]` generates the document's Table of Contents
- `@[]` or its alias `include[]` is used to copy the contents of another file stored anywhere in the `/text` directory.

Let's now analyze this `document.glyph` more in detail.

- The `document[]` macro wraps every other macro. This is necessary to create the initial `<html>` tag.
- Similarly, `head[]` and `body[]` are used to generate the respective HTML tags. Actually, `head[]` already sets some metadata for you, by default (author and copyright).
- Within `head[]`, the `style[]` macro is used to load the `default.css` stylesheet, which is included by default the `/styles` directory of every Glyph project.
- Immediately after the `body[]` macro, the `titlepage[]` macro is used to define (guess…) the first page of your document. `title[]`, `author[]` and `pubdate[]` insert the title of the document, its author and the publication date (retrieved from the project's configuration settings).

- Then, the `frontmatter[]`, `bodymatter[]` and `backmatter[]` macros are used to further divide the portions of your document according to the rules of book design. They are not mandatory, but they can be used, for example, to number your appendixes with letters instead of numbers and similar.
- `preface[]`, `chapter[]`, `appendix[]` are just a way to wrap content in `<div>` tags, from an HTML point of view, but they are also necessary to nest the content of your document and generate the Table of Contents automatically, together with the `header[]` macro.

1.3 Editing text files

1.4 Changing configuration settings

Glyph stores configuration settings in the following YAML files:

- Your *Project Configuration* is stored in the `config.yml` file, included in each Glyph Project.
- Your *Global Configuration* is stored in a `.glyphrc` file in your $HOME (or %HOMEPATH% on Windows) directory (not created by default).
- The *System Configuration* is stored in the source directory of Glyph itself.

When compiling, Glyph loads all these configuration files and merges them according to the following rules:

- A setting configured in the *Project Configuration* overrides the same setting in both Global and System configuration.
- A setting configured in the *Global Configuration* overrides the same setting in the *System Configuration*

Typically, you should use the *Project Configuration* for all project-specific settings and the *Global Configuration* for settings affecting all your projects (for example, you may want to set 'document.author' in the Global Configuration instead of setting it in the Project Configuration of all your Glyph projects). The *System Configuration* is best left untouched.

Instead of editing your configuration settings directly, you can use the `glyph config` command, as follows:

`glyph config` *setting [value]*

1.5 Compiling your project

Chapter II – Glyph Language

Glyph uses a simple macro language to perform advanced tasks like creating and validating links, generating Table of Contents and Indexes, and defining the document structure.

2.1 Introducing Glyph Macros

By now you probably figured out what a macro looks like: it's an identifier of some kind that wraps a value within square brackets. More specifically:

- The macro identifier can contain *any* character except for: `[`, `]`, `\`, `|` or spaces.
- The delimiters can be either `[` and `]` or `[=` and `=]` (for more information on differences between delimiters, see Escaping and Quoting).

2.2 Escaping and Quoting

2.3 Creating and Editing Macros

Appendix A – Command Reference

Appendix B – Macro Reference

Appendix C – Configuration Reference