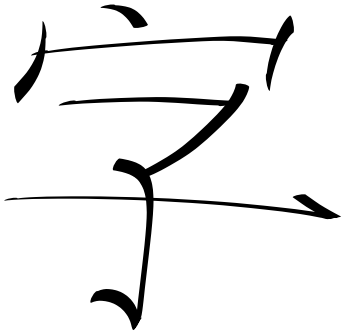

Glyph

Rapid Document Authoring Framework

v0.1.0

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 deliverables in different 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 – Getting Started

1.1 Creating your first Glyph Project

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` *test_document*

`cd` *test_document*

`glyph init`

That's it. You just created a new Glyph project in the `test_document` directory.

> **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)

Every Glyph project is comprised of the following directories:

- `images/` — used to store the image files used in your document.
- `lib/` — used to store your custom Glyph macros and Rake tasks.
- `output/` — used to store your generated output files.
- `styles/` — used to store your stylesheets.
- `text/*` — used to store your source text files.

Additionally, the following files are also created at top level:

- `config.yml` — containing your Project Configuration.
- `document.glyph` — containing your Document Structure
- `snippets.yml` — containing your text snippets.

1.2 Document Structure

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 Project Configuration

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]*

If no *value* is specified, glyph just prints the value of the configuration setting, so typing `glyph config document.author` right after creating a project (assuming you didn't set this in the Global Configuration) will print nothing, because this setting is blank by default.

To change the value of a configuration setting, specify a value right after the setting, like this:

```
glyph config document.author "John Smith"
```

In this way, the document author will be set to *John Smith* for the current project. To save this setting globally, add a `-g` option, like this:

```
glyph config -g document.author "John Smith"
```

Regarding configuration values and data types…

Glyph attempts to "guess" the data type of a configuration values by evaluation (`Kernel#instance_eval`) if the value:

- is wrapped in quotes (`"` or `'`) — String
- starts with a colon (`:`) — Symbol
- is wrapped in square brackets (`[` and `]`) — Array
- is wrapped in curly brackets (`\{` and `\}`) — Hash
- is *true* or *false* — Boolean
- If the value is *nil* — NilClass

Note that this guessing is far from being foolproof: If you type something like *{:test, 2}*, for example, you'll get an error.

There are plenty of configuration settings that can be modified, but most of them are best if left alone (and in the System Configuration file). For a complete reference, see Configuration Reference. Normally, you may just want to change the following ones:

Setting	Description
document.author	The author of the document
document.title	The title of the document
document.filename	The document file name

Chapter II – Authoring Documents

2.1 Text Editing

One of the aims of Glyph is streamlining text editing. Glyph accomplishes this through its own macro language that can be used in conjunction with Textile or Markdown.

2.1.1 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).

Macros can be nested within other macros: in this way it is possible to have sections within sections, links within headers and so on; actually, every Glyph document is contained within a single `document` macro.

2.1.2 Escaping and Quoting

Glyph doesn't require any special control characters like LaTeX, and its macro syntax is very straightforward and liberal. This however comes with a price: because square brackets are used as delimiters, you must escape any square bracket in your text with a backslash. That's not *too* bad if you think about it, unless you're writing programming code: in that case, escaping every single square bracket can be painful.

If a portion of your text contains an excessive amount of square brackets, you may consider using the `escape` macro (or better, its alias `.`) with `[=` and `=]` as delimiters. By itself, the escape macro doesn't do anything: it just evaluates to its contents, but the special delimiters act as a quote for any square bracket within them. As a consequence, any macro within `[=` and `=]` will *not* be evaluated.

You can use the quoting delimiters with *any* macro identifier. Obviously, using them as delimiters for things like `section` macros may not be a good idea, but they should really be mandatory with the `code` macro, like this:

```
code[=
section[header[A section]

This is a section.

        section[header[A nested section]
This is another section.
        ]
]
=]
```

> **Note**
> Although quoting delimiters allow you to use square brackets without escaping them, you must still escape them if you want to escape quoting delimiter themselves.

2.1.3 Sections and Headers

2.1.4 Including Files and Snippets

2.1.5 Links and Bookmarks

2.1.6 Images and Figures

2.2 Compiling your project

2.2.1 Adding Stylesheets

2.2.2 HTML output

2.2.3 PDF Output

Chapter III – Extending Glyph

3.1 Creating Glyph Macros

3.2 Creating Rake Tasks

Chapter IV – Troubleshooting

Appendix A – Command Reference

Appendix B – Macro Reference

Appendix C – Configuration Reference