XML workflow documentation

Carlos Araya

Table of Contents

Thank You

Thanks to Laura Brady for getting this particular idea started :)

Background: HTML is the final format

In researching the technologies and tools that I use when developing digital content I've come across multiple discussions about what's the best way to create HTML for X application (ebooks, web, transforming into other formats and any number of ideas. Some people think that HTML is perfect for everyone to write, regardless of experience and comfort with the technology. We forget that HTML now is very different to HTML as it was originally created.

> *HTML —which is short for HyperText Markup Language— is the official language of the World Wide Web and was first conceived in 1990. HTML is a product of SGML (Standard Generalized Markup Language) which is a complex, technical specification describing markup languages, especially those used in electronic document exchange, document management, and document publishing. HTML was originally created to allow those who were not specialized in SGML to publish and exchange scientific and other technical documents. HTML especially facilitated this exchange by incorporating the ability to link documents electronically using hyperlinks.*
>
> *From: http://www.ironspider.ca/webdesign101/htmlhistory.htm*

The biggest issue, in my opinion, is that HTML has become a lot more complicated than the initial design. Creating HTML content (particularly when used in conjunction with CSS frameworks like Bootstrap or Zurb or with applications that use additional semantic elements like ePub) takes a lot more than just knowing markup to code them correctly. It takes knowledge of the document structure, the semantics needed for the content or the applications we are creating and the restrictions and schemas that we need to use so that the content will pass validation.

This article presents 4 different approaches to creating HTML. Two of them use HTML directly but target it as the final output for transformations and templating engines; the other two use markup like HTML without requiring strict HTML conformance. I've made these selections for two reasons:

- People who are not profesionals should not have to learn all the details of creating an ePub3 table of content or know the classes to add to elements to create a Bootsrap or Foundation layout grid
- It makes it easier for developers and designers to build the layout for the content without having to worry about the content itself; we can play with layout and content organization in parallel with content creation and, if we need to make any further changes, we just run our compilation process again

Markdown

Perhaps the simplest solution when moving content from text to HTML is Markdown.

Markdownis a text to (X)HTML conversion tool designed for writers. It refers both to the syntax used in the Markdown text files and the applications used to perform the conversion.

Markdown language was created in 2004 by John Gruber with the goal of allowing people "to write using an easy-to-read, easy-to-write plain text format, and optionally convert it to structurally valid XHTML (or HTML)" (http://daringfireball.net/projects/markdown/)

The language was designed to be readable as-is, without all the additional tags and attributes that makes it possible to covert markdown to languages like SGML, XML and HTML. Markdown is a formatting syntax for text that can be read by humans and can be easily converted to HTML.

The original implementation of Markdown is markdown.pland has been implemented in several other languages as applications (Ruby Gems, NodeJS modules and Python packages). All versions of Markdown are distributed under open source licenses and are included or available as a plugin for, several content-management systems and text editors.

Sites such as GitHub, Reddit, Diaspora, Stack Overflow, OpenStreetMap, and SourceForge use variants of Markdown to facilitate content creation and discussion between users.

The biggest weakness of Markdown is the lack of a unified standard. The original Markdown language hasn't been really supported since it was released in 2004 and all new version of Markdown, both parser and language specification have introduced not wholy compatible changes to Markdown. The lack of standard is also Markdown's biggest strength. It means you can, like Github did, implement your own extensions to the Markdown syntax to acommodate your needs.

Markdown is not easy to learn but once your fingers get used to the way we type the different elements it becomes much easier to work with as it is nothing more than inserting specific characters in a specific order to obtain the desired effect. Once you train yourself, it is also easy to read without having to convert it to HTML or any other language.

Most modern text editors have support for Markdown either as part of the default installation or through plugins.

Example Markdown document

- Markdown example form daringfireball

Asciidoctor

I only discovered Asciidoctor recently, while researching O'Reilly Media's publishing toolchains. It caught my attention because of it's structure, the expresiveness of the markup without being HTML like HTMLbook and the extensibility of the templating system that it uses behind the scenes.

Asciidoctor has both a command line interface (CLI) and an API. The CLI is a drop-in replacement for the *asciidoc* command from the Standard python distribution. This means that you have a command line tool *asciidoctor* that will allow you to convert your marked documents without having to resort to a full blown application.

Syntax-wise, Asciidoctor is progressively more complex as you implement more advanced features. In the first example below no tables are used, for example. Tables are used in the second and thirs examples both as data tables and for layout.

The http://asciidoctor.org/docs/ provides more detailed instructions for the desired markup.

Example Asciidoc documents

- Asciidoctor planning document
- Comparison of Asciidoctor and AsciiDoc Features
- Applying Custom Themes

HTMLBook

O'Reilly Media has developed several new tools to get content from authors to readers. Atlas is their authoring tool, a web based application that allows you to create content they developed HTMLbook, a subset of HTML geared towards authoring and multi format publishing.

Given O'Reilly's history and association with open source publishing tools (they were an early adopter and promoter of Docbook and still use it for some of their publications) I found HTMLbook intriguing but not something to look at right away, as with many things you leave for later it fell off my radar.

It wasn't until I saw Sanders Kleinfeld's (O'Reilly Media Director of Publishing Technologies) presentation at IDPF Book World conference that I decided to take a second look at HTMLbook and its ecosystem.

Conceptually HTMLbook is very simple; it combines a subset of HTML5, the semantic structure of ePub documents and other IDPF specifications to create a flavor of HTML 5 that is designed specifically for publishing. There are also stylesheets that will allow you to convert Markdown and other text formats into HTMLbook (see Markdown to HTMLBook and AsciiDoc to HTMLBook (via AsciiDoctor))

If you use Atlas (O'Reilly's authoring and publishing platform) you don't have to worry about markup as the content is created visually. The challenges begin when implementing this vocabulary outside the Atlas environment.

The project comes with a set of stylesheets to convert HTMLbook content to ePub, MOBI and PDF. The intriguing thing about the stylesheets is that they use CSS Paged Media stylesheets in conjunction with third party tools such as AntennaHouse or PrinceXML.

The open source solutions offer permisive licenses that allow modification and integration into other products without requiring you to release your project under the same license like GPL and LGPL.

As with any solution that advocates creating HTML directly I have my reservations. In HTML formating in general and specialized formats like HTMLbooks in particular, the learning curve may be too steep for independent authors to use for creating content.

The user must learn not only the required HTML5 syntax but also the details regarding ePub semantic structure attributes and the other standards needed to create ePub books. While I understand that technologies such as this are not meant for independent authors or for poeple who are not comfortable or familiar with HTML but the learning curve may still be too steep for most users.

Example HTMLbook document

- Alice Adventures in Wonderland marked as HTMLbook

XML / XSLT

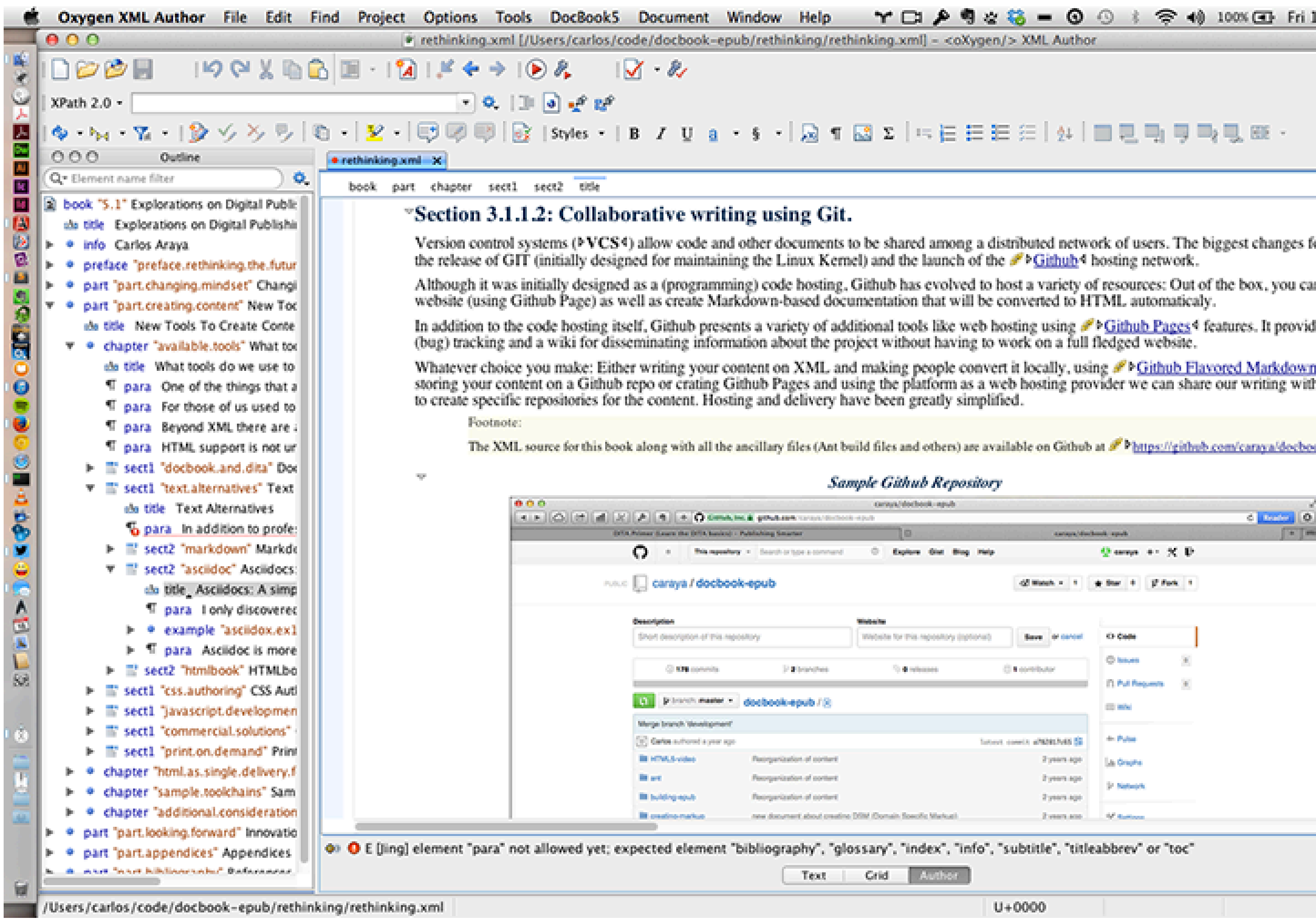
Perhaps the oldest solutions in the book to create HTML without actually creating HTML are XML-based. Docbook, TEI and DITA all have stylesheets that will take the XML content and convert it to HTML, PDF, ePub and other more esoteric formats.

In addition to stylesheets already available developers can create their own to adress specific needs.

Furthermore, tools like OxygenXML Author (and I would assume other tools in the same category have a visual mode that allow users to write XML content, validated against a schema in a way that is more familiar to people not used to creating content with raw XML tools.

The issues with xml are similar to those involved in creating HTML. The markup vocabulary requires brackets, attributes have to be enclosed in quotation marks and generall the syntax is as complicated as you make it. However, tools like Oxygen and smilar help alleviate this problem but don't resolve it completely.

The screenshot below shows OxygenXML Author working in a Docbook 5 document using visual mode.


Oxygen XML Author File Edit Find Project Options Tools DocBook5 Document Window Help
rethinking.xml [/Users/carlos/code/docbook-epub/rethinking/rethinking.xml] - <oXygen/> XML Author
XPath 2.0
Styles
Outline
Element name filter
rethinking.xml
book part chapter sect1 sect2 title
Text Grid Author
U+0000

book "5.1" Explorations on Digital Publi
title Explorations on Digital Publishi
info Carlos Araya
preface "preface.rethinking.the.futur
part "part.changing.mindset" Changi
part "part.creating.content" New Too
title New Tools To Create Conte
chapter "available.tools" What too
title What tools do we use to
para One of the things that a
para For those of us used to
para Beyond XML there are
para HTML support is not ur
sect1 "docbook.and.dita" Doc
sect1 "text.alternatives" Text
title Text Alternatives
para In addition to profe
sect2 "markdown" Markd
sect2 "asciidoc" Asciidocs
title Asciidocs: A simp
para I only discovered
example "ascidox.ex1
para Ascidoc is more
sect2 "htmlbook" HTMLbo
sect1 "css.authoring" CSS Aut
sect1 "javascript.developmen
sect1 "commercial.solutions"
sect1 "print.on.demand" Prin
chapter "html.as.single.delivery.f
chapter "sample.toolchains" Sam
chapter "additional.consideration
part "part.looking.forward" Innovatio
part "part.appendices" Appendices

Section 3.1.1.2: Collaborative writing using Git.

Version control systems (VCS) allow code and other documents to be shared among a distributed network of users. The biggest changes fo the release of GIT (initially designed for maintaining the Linux Kernel) and the launch of the Github hosting network.

Although it was initially designed as a (programming) code hosting, Github has evolved to host a variety of resources: Out of the box, you ca website (using Github Page) as well as create Markdown-based documentation that will be converted to HTML automaticaly.

In addition to the code hosting itself, Github presents a variety of additional tools like web hosting using Github Pages features. It provid (bug) tracking and a wiki for disseminating information about the project without having to work on a full fledged website.

Whatever choice you make: Either writing your content on XML and making people convert it locally, using Github Flavored Markdown storing your content on a Github repo or crating Github Pages and using the platform as a web hosting provider we can share our writing wit to create specific repositories for the content. Hosting and delivery have been greatly simplified.

Footnote:

The XML source for this book along with all the ancillary files (Ant build files and others) are available on Github at https://github.com/caraya/docbo

Sample Github Repository

E [Jing] element "para" not allowed yet; expected element "bibliography", "glossary", "index", "info", "subtitle", "titleabbrev" or "toc"

/Users/carlos/code/docbook-epub/rethinking/rethinking.xml

Figure 1: OxygenXML Visual Editor for XML

The positive side is that using XSLT there is no limit to what we can do with our XML content.

XML examples

- Sample Chapter marked as Docbook
- Tales by Edgar Allan Poe, marked as TEI
- Chapter from 20000 Leagues Under the Sea, marked as DITA

Conclusion

After exploring a selection of HTML conversion options the question becomes *which one is best?*

The answer is *it depends*.

The best way to see how can these text-based tools can be incorporated is to ask yourself how much work you want to do in the backend versus how much work do you want you authors to do when creating the content. This is where the value of specialists in digital formats and publishing becomes essential, we can work with clients in providing the best solution to meet their needs.

Keep in mind who your audience and what the target vocabulary you're working towards, it will dictate what your best strategy is. Are these all the solutions; definitely not. Other solutions may appear that fit your needs better than those presented here; I would love to hear if that is the case.

Striking the balance between author and publisher is a delicate one. I tend to fall on the side of making things easier for authors... The tools can be made to translate basic markup into the desired result with minimal requirements for authors to mark up the content; the same can't necessarily be said about the publisher-first strategy

Background: HTML is the final product

In researching the technologies and tools that I use when developing digital content I've come across multiple discussions about what's the best way to create HTML for X application (ebooks, web, transforming into other formats and any number of ideas. Some people think that HTML is perfect for everyone to write, regardless of experience and comfort with the technology. We forget that HTML now is very different to HTML as it was originally created.

> *HTML —which is short for HyperText Markup Language— is the official language of the World Wide Web and was first conceived in 1990. HTML is a product of SGML (Standard Generalized Markup Language) which is a complex, technical specification describing markup languages, especially those used in electronic document exchange, document management, and document publishing. HTML was originally created to allow those who were not specialized in SGML to publish and exchange scientific and other technical documents. HTML especially facilitated this exchange by incorporating the ability to link documents electronically using hyperlinks.*
>
> *From http://www.ironspider.ca/webdesign101/htmlhistory.htm*

The biggest issue, in my opinion, is that HTML has become a lot more complicated than the initial design. Creating HTML content (particularly when used in conjunction with CSS frameworks like Bootstrap or Zurb or with applications that use additional semantic elements like ePub) takes a lot more than just knowing markup to code them correctly. It takes knowledge of the document structure, the semantics needed for the content or the applications we are creating and the restrictions and schemas that we need to use so that the content will pass validation.

This article presents 4 different approaches to creating HTML. Two of them use HTML directly but target it as the final output for transformations and templating engines; the other two use markup like HTML without requiring strict HTML conformance. I've made these selections for two reasons:

- People who are not profesionals should not have to learn all the details of creating an ePub3 table of content or know the classes to add to elements to create a Bootsrap or Foundation layout grid
- It makes it easier for developers and designers to build the layout for the content without having to worry about the content itself; we can play with layout and content organization in parallel with content creation and, if we need to make any further changes, we just run our compilation process again

Markdown

Perhaps the simplest solution when moving content from text to HTML is Markdown.

Markdown is a text to (X)HTML conversion tool designed for writers. It refers both to the syntax used in the Markdown text files and the applications used to perform the conversion.

Markdown language was created in 2004 by John Gruber with the goal of allowing people "to write using an easy-to-read, easy-to-write plain text format, and optionally convert it to structurally valid XHTML (or HTML)" (http://daringfireball.net/projects/markdown/)

The language was designed to be readable as-is, without all the additional tags and attributes that makes it possible to covert markdown to languages like SGML, XML and HTML. Markdown is a formatting syntax for text that can be read by humans and can be easily converted to HTML.

The original implementation of Markdown is markdown.pl and has been implemented in several other languages as applications (Ruby Gems, NodeJS modules and Python packages). All versions of Markdown are distributed under open source licenses and are included or available as a plugin for, several content-management systems and text editors.

Sites such as GitHub, Reddit, Diaspora, Stack Overflow, OpenStreetMap, and SourceForge use variants of Markdown to facilitate content creation and discussion between users.

The biggest weakness of Markdown is the lack of a unified standard. The original Markdown language hasn't been really supported since it was released in 2004 and all new version of Markdown, both parser and language specification have introduced not wholy compatible changes to Markdown. The lack of standard is also Markdown's biggest strength. It means you can, like Github did, implement your own extensions to the Markdown syntax to acommodate your needs.

Markdown is not easy to learn but once your fingers get used to the way we type the different elements it becomes much easier to work with as it is nothing more than inserting specific characters in a specific order to obtain the desired effect. Once you train yourself, it is also easy to read without having to convert it to HTML or any other language.

Most modern text editors have support for Markdown either as part of the default installation or through plugins.

Example Markdown document

- Markdown example form daringfireball

Asciidoctor

I only discovered Asciidoctor recently, while researching O'Reilly Media's publishing toolchains. It caught my attention because of it's structure, the expresiveness of the markup without being HTML like HTMLbook and the extensibility of the templating system that it uses behind the scenes.

Asciidoctor has both a command line interface (CLI) and an API. The CLI is a drop-in replacement for the *asciidoc* command from the Standard python distribution. This means that you have a command line tool *asciidoctor* that will allow you to convert your marked documents without having to resort to a full blown application.

Syntax-wise, Asciidoctor is progressively more complex as you implement more advanced features. In the first example below no tables are used, for example. Tables are used in the second and thirs examples both as data tables and for layout.

The documentation provides more detailed instructions for the desired markup.

Example Asciidoc documents

- Asciidoctor planning document
- Comparison of Asciidoctor and AsciiDoc Features
- Applying Custom Themes

HTMLBook

O'Reilly Media has developed several new tools to get content from authors to readers. Atlas is their authoring tool, a web based application that allows you to create content they developed HTMLbook, a subset of HTML geared towards authoring and multi format publishing.

Given O'Reilly's history and association with open source publishing tools (they were an early adopter and promoter of Docbook and still use it for some of their publications) I found HTMLbook intriguing but not something to look at right away, as with many things you leave for later it fell off my radar.

It wasn't until I saw Sanders Kleinfeld's (O'Reilly Media Director of Publishing Technologies) [presentation at IDPF Book World conference](http://www.slideshare.net/sanderskleinfeld/open-source-forpubsslideshare) that I decided to take a second look at HTMLbook and its ecosystem.

Conceptually HTMLbook is very simple; it combines a subset of HTML5, the semantic structure of ePub documents and other IDPF specifications to create a flavor of HTML 5 that is designed specifically for publishing. There are also stylesheets that will allow you to convert Markdown and other text formats into HTMLbook (see Markdown to HTMLBook) and https://github.com/oreillymedia/asciidoctor-htmlbook/.)

If you use Atlas (O'Reilly's authoring and publishing platform) you don't have to worry about markup as the content is created visually. The challenges begin when implementing this vocabulary outside the Atlas environment.

The project comes with a set of stylesheets to convert HTMLbook content to ePub, MOBI and PDF. The intriguing thing about the stylesheets is that they use CSS Paged Media stylesheets in conjunction with third party tools such as AntennaHouse or PrinceXML.

As with any solution that advocates creating HTML directly I have my reservations. In HTML formating in general and specialized formats like HTMLbooks in particular, the learning curve may be too steep for independent authors to use for creating content.

The user must learn not only the required HTML5 syntax but also the details regarding ePub semantic structure attributes and the other standards needed to create ePub books. While I understand that technologies such as this are not meant for independent authors or for poeple who are not comfortable or familiar with HTML but the learning curve may still be too steep for most users.

Example HTMLbook document

- Alice Adventures in Wonderland marked as HTMLbook]

XML / XSLT

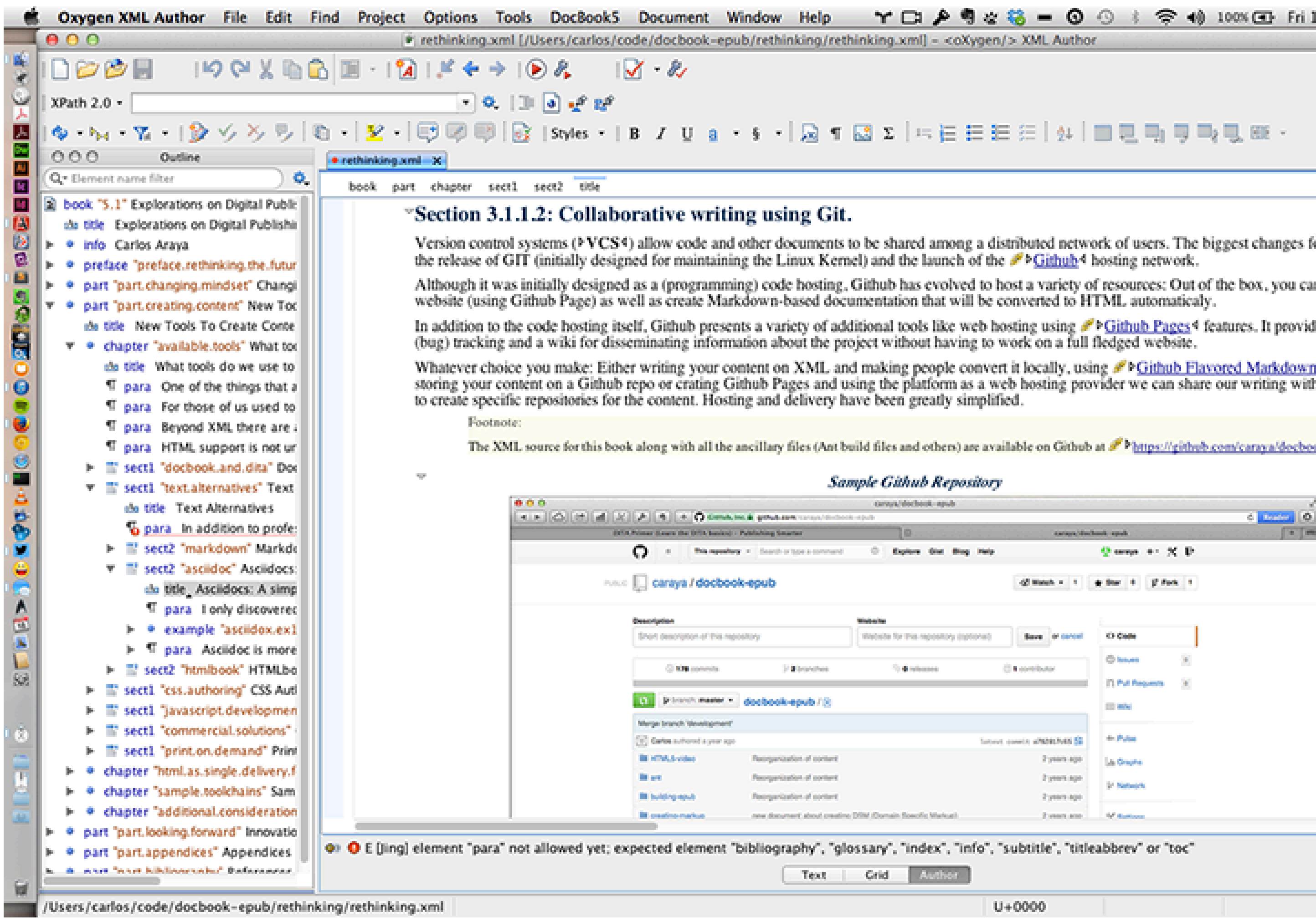
Perhaps the oldest solutions in the book to create HTML without actually creating HTML are XML-based. Docbook, TEI and DITA all have stylesheets that will take the XML content and convert it to HTML, PDF, ePub and other more esoteric formats.

In addition to stylesheets already available developers can create their own to adress specific needs.

Furthermore, tools like OxygenXML Author (and I would assume other tools in the same category have a visual mode that allow users to write XML content, validated against a schema in a way that is more familiar to people not used to creating content with raw XML tools.

The issues with xml are similar to those involved in creating HTML. The markup vocabulary requires brackets, attributes have to be enclosed in quotation marks and generall the syntax is as complicated as you make it. However, tools like Oxygen and smilar help alleviate this problem but don't resolve it completely.

The screenshot below shows OxygenXML Author working in a Docbook 5 document using visual mode.


Oxygen XML Author File Edit Find Project Options Tools DocBook5 Document Window Help
rethinking.xml [/Users/carlos/code/docbook-epub/rethinking/rethinking.xml] - <oXygen/> XML Author
XPath 2.0
Styles
Outline
Element name filter
rethinking.xml
book part chapter sect1 sect2 title
Text Grid Author
U+0000

```
book "5.1" Explorations on Digital Publi
  title Explorations on Digital Publishi
  info Carlos Araya
  preface "preface.rethinking.the.futur
  part "part.changing.mindset" Changi
  part "part.creating.content" New Too
    title New Tools To Create Conte
    chapter "available.tools" What too
      title What tools do we use to
      para One of the things that a
      para For those of us used to
      para Beyond XML there are
      para HTML support is not ur
      sect1 "docbook.and.dita" Doc
      sect1 "text.alternatives" Text
        title Text Alternatives
        para In addition to profe
        sect2 "markdown" Markdo
        sect2 "asciidoc" Asciidocs:
          title Asciidocs: A simp
          para I only discovered
          example "asciidox.ex1
          para Asciidoc is more
        sect2 "htmlbook" HTMLbo
      sect1 "css.authoring" CSS Autl
      sect1 "javascript.developmen
      sect1 "commercial.solutions"
      sect1 "print.on.demand" Print
    chapter "html.as.single.delivery.f
    chapter "sample.toolchains" Sam
    chapter "additional.consideration
  part "part.looking.forward" Innovatio
  part "part.appendices" Appendices
```

Section 3.1.1.2: Collaborative writing using Git.

Version control systems (VCS) allow code and other documents to be shared among a distributed network of users. The biggest changes f
the release of GIT (initially designed for maintaining the Linux Kernel) and the launch of the Github hosting network.

Although it was initially designed as a (programming) code hosting, Github has evolved to host a variety of resources: Out of the box, you ca
website (using Github Page) as well as create Markdown-based documentation that will be converted to HTML automaticaly.

In addition to the code hosting itself, Github presents a variety of additional tools like web hosting using Github Pages features. It provid
(bug) tracking and a wiki for disseminating information about the project without having to work on a full fledged website.

Whatever choice you make: Either writing your content on XML and making people convert it locally, using Github Flavored Markdown
storing your content on a Github repo or crating Github Pages and using the platform as a web hosting provider we can share our writing wit
to create specific repositories for the content. Hosting and delivery have been greatly simplified.

Footnote:

The XML source for this book along with all the ancillary files (Ant build files and others) are available on Github at https://github.com/caraya/docbo

Sample Github Repository

E [Jing] element "para" not allowed yet; expected element "bibliography", "glossary", "index", "info", "subtitle", "titleabbrev" or "toc"

/Users/carlos/code/docbook-epub/rethinking/rethinking.xml

The positive side is that using XSLT there is no limit to what we can do with our XML content.

XML examples

- [Sample Chapter marked as Docbook]()
- [Tales by Edgar Allan Poe, marked as TEI]()
- [Chapter from 20000 Leagues Under the Sea, marked as DITA]()

Conclusion

After exploring a selection of HTML conversion options the question becomes *which one is best?*

The answer is *it depends.*

The best way to see how can these text-based tools can be incorporated is to ask yourself how much work you want to do in the backend versus how much work do you want you authors to do when creating the content. This is where the value of specialists in digital formats and publishing becomes essential, we can work with clients in providing the best solution to meet their needs.

Keep in mind who your audience and what the target vocabulary you're working towards, it will dictate what your best strategy is. Are these all the solutions; definitely not. Other solutions may appear that fit your needs better than those presented here; I would love to hear if that is the case.

Striking the balance between author and publisher is a delicate one. I tend to fall on the side of making things easier for authors... The tools can be made to translate basic markup into the desired result with minimal requirements for authors to mark up the content; the same can't necessarily be said about the publisher-first strategy

Introduction

One of the biggest limitations of markup languages, in my opinion, is how confining they are. Even large vocabularies like Docbook are limited in what they can do out of the box. HTML4 is non-extensible and HTML5 is limited in how you can extend it (web components are the only way to extend HTML5 I'm aware of that doesn't require an update to the HTML specification.)

By creating our own markup vocabulary we can be as expressive as we need to be without adding additional complexity for writers and users and without adding unecessary complexity for the developers building the tools to interact with the markup.

Why create our own markup

I have a few answers to that question:

In creating your own xml-based markup you enforce separation of content and style. The XML document provides the basic content of the document and the hints to use elsewhere. XSLT stylesheets allow you to structure the base document and associated hints into any number of formats (for the purposes of this document we'll concentrate on XHTML, PDF created through Paged Media CSS and ePub)

Creating a domain specific markup vocabulary allows you think about structure and complexity for yourself as the editor/typesetter and for your authors. It makes you think about elements and attributes and which one is better for the given experience you want and what, if any, restrictions you want to impose on your makeup.

By creating our own vocabulary we make it easier for authors to write clean and simple content. XML provides a host of validation tools to enforce the structure and format of the XML document.

Options for defining the markup

For the purpose of this project we'll define a set of resources that work with a book structure like the one below:

```
<book>
<metadata>
<title>The adventures of SHerlock Holmes</title>

<author>
<first-name>Arthur</first-name>
<surname>Connan Doyle</surname>
</author>
</metadata>

<section type="chapter">
<para>Lorem Ipsum</para>
<para>Lorem Ipsum</para>
</section>
</book>
```

It is not a complete structure. We will continue adding elements afte we reach the MVP (Minimum Viable Product) stage. As usual, feedback is always appreciated.