Making ebooks with Sigil, HTML and CSS

Training notes for book designers and editors

Arthur Attwell

EBW

EBW

Making ebooks with Sigil, HTML and CSS

- First published by Electric Book Works on 4 March 2014
- Revised 9 October 2014

ISBN: 978-1-920218-80-5

Contents

Agenda 6
1 HTML, or ‘How to Talk a Machine’s Language’ 8
2 Stylesheets 14
3 Working with images 23
4 Links 26
5 Metadata 30
6 Tables of Contents 33
7 Covers 35
8 Testing and validation 37
9 On advanced tricks and bug fixing 39
10 Your feedback 40

Course outline

In the course, we cover the following themes in detail, all in the context of using Sigil to create ebooks.

- An introduction to opening and reading ebooks in ereaders
- An intro to HTML
- Matching HTML to parts of books
- Formatting with CSS
- Working with images
- Links (internal and external)
- Metadata
- Tables of Contents
- Covers
- Testing and validation
- Advanced tricks and bug fixing

Not all of these things are covered in these notes, but you can always read more just by searching online. No matter how many ebooks you make, you'll always hit snags, and searching online is always the fastest way to find answers. We've written about many on the Electric Book Works Knowledge Base at electricbookworks.com/kb.

Finally, please give us your feedback after the course. You can use the feedback form in these notes, or just pop us a mail at info@electricbookworks.com.

HTML, or ‘How to Talk a Machine’s Language’

HTML actually stands for ‘hypertext markup language’, and you’ll understand why soon. When you make an ebook you are talking to a machine. That is, you’re making a document that a computer will understand.

Look at this:

> Hello, World!

What do we humans call this? Words, a sentence, exclamation, line, phrase, paragraph, letters, sign, greeting – we have dozens of names for it, and that’s just if we’re speaking English. To agree on what it is, we need:

- a language (English)
- common terms (like the ones I listed above)

When we tell a computer what Hello, World! is, we also need a language and common terms. In the ebook world:

- the language is HTML
- common terms include ‘paragraph’, ‘string’, and ‘span’.

To indicate that common term to a computer, we surround our phrase with special HTML tags. Tags always appear in elbow brackets, like this:

```
<p>Hello World!</p>
```

The `<p>` is a ‘paragraph’ tag. We use paragraph tags at the start and end of the paragraph, and the slash indicates that the tag is closing the paragraph. These tags are called ‘markup’, because they mark up our text for the computer.

Elements

In HTML terms, a paragraph marked up with `<p>` tags is an 'element'. The word 'element' would also be useful in traditional book terms for any piece of a book, and for what we often call 'features'.

HTML includes a bunch of standard elements. You'll get to know them very well, especially these ones:

- `<p>` for paragraph
- `<ul>` for unordered list (i.e. a bulleted list, but it might not actually have visible bullets)
- `<ol>` for ordered list (e.g. a 1, 2, 3 or a, b, c list)
- `<li>` for list item, an item in an ordered or unordered list
- `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>` for six levels of heading
- `<img>` for image
- `<a>` for a clickable link (the 'a' happens to stand for anchor)
- `<em>` for emphasising words (as you might use italics in print)
- `<strong>` for making words stand out (as you might use bold in print)
- `<span>` for any string of letters you need to mark for any reason (as in, this span `<span>spans three words</span>`)
- `<div>` for division, any block of text or images you need to mark for any reason
- `<table>` for a table
- `<tr>` for a table row
- `<td>` for a table cell in a row (`td` stands for table data).

You must know these elements to make ebooks. There are others that are less important, and you'll learn about them along the way.

Classes

As you may have guessed, in most books there are several different kinds of paragraph. In addition to plain body text, there are pull quotes, chapter openers, notes and more.

In HTML we say there are different *classes* of paragraph. In theory, there are as many different paragraph classes as you could ever think of. The same goes for any of the elements I listed:

- lists might be regular bulleted lists, clickable menus, chapter objectives, or glossary entries;
- images might be portraits, graphs, icons, decorations, or maps;
- spans might mark computer code, important words, proper names, places, or other special terms;
- divs might mark sections as sidebars, footnotes, or activities.

And much more. We can't just use `<p>` for everything, because then every class of paragraph would look the same.

So HTML lets us invent our own classes, and gives us a way to say what class an element belongs to. We do this by adding an *attribute* to the element's opening tag. Let's say we want to call our paragraph a 'greeting' paragraph:

```
<p class="greeting">Hello World!</p>
```

We could have called the class anything we liked:

```
<p class="frabjous-day">Hello World!</p>
```

But of course it's best to use class names that are easy to remember and describe their purpose clearly.

ATTRIBUTE STRUCTURE

An attribute in a tag always takes the same form: the kind of attribute (e.g. class), an equals sign, and the attribute value in quote marks. Another kind of attribute is an ID, which we cover in Links, later.)

> If you're sharp, you'll have noticed that none of these elements or classes describe what the text or image they mark up *looks like*. They only describe its *function or purpose.*

All appearance or formatting is managed separately, in another file called a stylesheet. We'll get to that later.

IDs

Elements can have IDs as attributes, too. An ID identifies that particular element uniquely.

```
<p id="jiminycricket">This paragraph is the only
one about Jiminy Cricket.</p>
```

IDs are especially useful when we want to link to specific elements, like a given paragraph or image. We cover this in the ‘Links’ chapter.

HTML document structure

If you’re using Sigil, you may know already that it lets you view your book in Book View or Code View. In Book View, things look a lot like a word-processor: what you type is what your end-user will see. What You See Is What You Get: WYSIWYG. In Code View, you see all the tags we’ve been talking about. We say that an ereader *renders* the code in its intended form.

Now that you know about tags, we can talk about how they’re structured. Their structure must follow certain rules.

An ebook might contain several HTML documents. Every HTML document is a separate text file with at least these four parts:

- a namespace declaration
- an `<html>` element
- a `<head>` element
- a `<body>` element.

Don’t worry about the namespace declaration. Sigil puts it in for you at the top of every file. It’s two or three lines long, and starts with `<?xml`. This is important info for the computer, but it’s rare that you’ll need to change it or even look at it closely.

The `<html>`, `<head>` and `<body>` elements always appear in this structure:

```
 <html>
        <head></head>
        <body></body>
<html>
```

That is, the whole document is wrapped inside the `<html>` element. And that's divided into two sections: the `<head>` element and the `<body>` element.

- The `<head>` contains information for the device about the document.
- The `<body>` contains all the content of the document intended for human readers.

There is only ever one `<html>`, one `<head>` and one `<body>` element. All the tags we discussed earlier (paragraphs, spans, divs, lists, etc.) go inside the `<body>` element to make up the document's content. When you're editing ebooks, 99 per cent of your time will be spent there, inside the `<body>` element.

XHTML

If you're looking around in Sigil, you might have noticed that files in most epubs are in XHTML, not HTML (see the file extensions in the Text folder). XHTML is essentially a stricter form of HTML, but for our purposes, it's the same thing.

Stylesheets

In your HTML documents, your job is to mark up text and images according to its function or purpose. Never its appearance. All the appearance work is done in a separate file called a stylesheet. In the stylesheet, for instance, you get to say what a heading marked `<h1>` looks like: colour, size, font, spacing, and so on. This way, you can change and control the appearance of all the elements in your book in one place, keeping formatting consistent everywhere.

We create stylesheets in a language called CSS, for Cascading Style Sheets. (Don't worry now about how they cascade: that'll become clear later.) In Sigil, HTML documents are stored in the Text folder of your epub, and CSS documents in the Styles folder.

CSS looks different to HTML, because it's a different language with its own syntax. Let's say you want to control the appearance of your default paragraphs. In CSS, you'd start by typing:

`p {}`

The `p` refers to the HTML `<p>` element, obviously. All your formatting instructions will go inside the braces, `{}`. For instance:

```
p {
        font-size: 12pt;
        font-family: serif;
        }
```

I've put each instruction on its own, new, indented line to make it easier for us humans to read – the computer doesn't mind. It's the same as writing this:

`p {font-size: 12pt; font-family: serif;}`

But when CSS documents get long and complicated, you really want things laid out neatly so that you can edit them easily.

Now, what about our classes? Like the 'greeting' class we created in `<p class="greeting">Hello World!</p>`? In CSS, we use a full-stop to indicate 'in class', like this:

```
p.greeting {
        color: pink;
        }
```

Once we've added that, our `Hello, World!` will show in pink in our ebook. (Note the American spelling of `color`. The same goes for instructions like `text-align: center;`.)

Open this ebook in Sigil and take a look at the CSS file to see a longer, but still very simple stylesheet. Note that you can add comments for humans that the computer will ignore by putting them between `/*` and `*/`.

Linking HTML and CSS files

Once you've created a CSS file, you have to tell your HTML file where to find it. That is, you have to *link* your HTML to the CSS file.

We do this by adding a `<link>` element inside the `<head>` section of the HTML document. Let's say your CSS file is called `styles.css`. In Sigil, your `<link>` tag will look like this:

`<link href="../Styles/styles.css" rel="stylesheet" type="text/css" />`

This link element contains three attributes:

1. `<href=…>` (`href` for 'hyperlink reference') says where the stylesheet is, relative to the HTML document (the two full-tops mean 'go up out of the Text folder', and the `/Styles/styles.css` means 'go into the Styles folder and find the `styles.css` file);
2. `rel="stylesheet"` says that the file's relationship to the HTML is that of a stylesheet;
3. `type="text/css"` says that the file will be in plain text and in the CSS language.

NOTE: SELF-CLOSING TAGS

The link element needs only one tag, because it doesn't mark up, or surround, any text. So it *closes itself* with a 'self-closing slash' at the end of the tag. Two other common HTML elements that close themselves are the line break `<br />` and the image tag `<img />`.

Challenge

Here are several things you might want to do with CSS. Pick one, get onto Google, and take 5 minutes to have a guess how you'd write the CSS:

- make the background of your whole ebook yellow;
- put a paragraph in a box;
- make a paragraph float to one side, with the body text flowing around it;
- make one word appear in a different font to the surrounding text;
- make a heading appear in uppercase, without changing the letters to uppercase in your HTML;
- create a list inside a list (a sub-list);
- make a numbered list number in roman numerals.

Sample stylesheet

Here is a sample stylesheet. These are styles used for the on-screen version of these notes. I've only included the parts that relate to ebook typography, but you can see the entire file online at electricbookworks.github.io/ebw-training. Open the page, right click to view the page source, and click on the CSS file name.

```
 body {
        font-family: "Source Sans Pro", serif;
        font-size: 1.2em;
        font-weight: 300;
        line-height: 140%;
}

/* Headings */

h1, h2, h3, h4, h5, h6 {
        font-weight: 600;
        line-height: 120%;
        page-break-after: avoid;
}
h1 {
        font-size: 3.5em;
        max-width: 80%;
        line-height: 100%;
}
h2 {
        margin: 1.5em 0 0.5em 0;
        padding-top: 0.5em;
}
h3 {
        margin: 1.5em 0 0.5em 0;
        padding-top: 0.5em;
}

/* Paragraphs and related */

p, ul, ol, blockquote, dl {
        max-width: 40em;
}
p {
        margin: 0 0 0.5em 0;
}
li p { /* a paragraph inside a list item */
        margin: 0;
}

/* Blockquotes */

blockquote {
        margin: 0.5em 0 0.5em 0;
        padding: 0.5em 1em 0.5em 1em;
        font-size: 1.5em;
        line-height: 120%;
        border-left: 5px solid #ddd;
        max-width: 20em;
}
blockquote p { /* a paragraph inside a blockquote */
        margin: 0;
        padding: 0;
}
li blockquote { /* a list item inside a blockquote
*/
        margin: 1em 0;
}

/* Definition lists, which I use for side-notes */

dl {
        color: #666;
        border: 1px solid grey;
        padding: 1em;
}
dt {
        margin: 0 0 0.25em 0;
        font-weight: 600;
}
dd {
}

/* Tables */

table {
        border-collapse: collapse;
}
th, td {
        border: 1px solid #ccc;
        padding: 0.5em;
}

/* Code text */
```

```
pre, code {
        font-family: "Source Code Pro", monospace;
        white-space: pre-wrap;
        background-color: #f9f9f9;
        padding: 0.1em 0.3em;
        border-radius: 0.2em;
        font-weight: 200;
        font-size: 0.9em;
}

/* Links */

a {
        text-decoration: none;
        color: #5f738c;
}

/*
 * Title page
 */

.titlepage-title {
        font-size: 3em;
        font-weight: 700;
        margin: 20% 0 0 0;
        line-height: 100%;
}
.titlepage-subtitle {
        font-size: 3em;
        font-weight: 400;
        margin: 0 0 1em 0;
        line-height: 100%;
}
.titlepage-author {
        font-size: 1.5em;
        margin: 0 0 2em 0;
```

```
}
.titlepage-logo {
        width: 100px;
}

/*
 * Copyright page
 */

.copyright-logo {
        width: 100px;
        margin: 1em 0;
}

/*
 * Table of contents
 */

.toc h2 {
        font-weight: 400;
        font-size: 1.25em;
        margin: 1em 0 0.5em 0;
}
.toc ol {
        margin: 0;
        padding: 0;
}
.toc ul {
        list-style-type: none;
        margin: 0 0 1em 0;
        padding: 0;
}
.toc li {
        margin: 0 0 0.5em 0;
}
.toc a {
        text-decoration: none;
}
```

Working with images

You are going to want to add or edit images in ebooks often. Luckily, this is very simple. You can just use Sigil's `Insert > Files…` menu. But if you want to understand what's going on in your code, here is the more manual method. To troubleshoot, you'll need to know this stuff.

Add the image file

First, add the image to the epub's `Images` folder. In Sigil, just right-click the `Images` folder and select `Add Existing Files`. Then go find the image on your computer, and Sigil will copy it to that `Images` folder. Remember, though:

- remove all spaces from image file names;
- try to use only lowercase letters in file names (for one thing, Sigil alphabetises file names differently according to the case of the letters);
- only use jpg or gif images (png is usually also okay);
- before adding, make sure the images are 72 to 150 dpi, large but not larger than 1000px on their longest side, and saved in RGB colours, not CMYK.

Add an image tag

Now, head back to your HTML document to add an `<img>` image tag where the image must appear. Image tags must always include two attributes:

- `src` for source, indicating the image file you just added, and
- `alt` for alternative text, which appears if the image breaks and is read out loud is reader for the visually impaired.

An image tag can also include a `title` attribute, which provides more info about the image. Many ereaders show the title as a tooltip when a user mouses over the image.

Finally, the image tag must self-close with a slash.

Here's an example:

```
<img src="../Images/example.jpg" alt="Example
image" title="An example of an image in an ebook"
/>
```

Done! Switch back to book view to check that it's displaying correctly.

Image sizes and styles

Images vary in size and are portrait or landscape or square. So are the screens they appear on. This means you will need to experiment with CSS styling for your images to make sure they display reliably on various screen sizes and orientations. Usually, this means working with `max-width` and `max-height` rules in your CSS.

Sometimes you'll need to create a couple of classes for different kinds of images in your book. For instance, little marginal decorations might be in a `margin-decor` class, while big, important graphics might be in an `important-graphic` class. Remember to create class names that describe the *purpose* of a given kind of image, not its appearance.

Captions and figures

If your images have captions, you should have two options:

- Make the captions paragraphs immediately before or after the image (you can make them `<p class="caption">` to style them); or
- Use the HTML for a `<figure>` element.

But the second option, using `<figure>` doesn't work in Sigil. Still, it's useful to know about it, in case Sigil allows for it in future, or in the event that you use another epub editor that does allow it.

The `<figure>` element wraps an image and its caption together. This is useful for styling, and also for keeping the image and its caption, marked up with `<figcaption>`, on the same page. The HTML for a figure looks like this:

```
<figure>
    <img src="../Images/myimage.jpg" alt="My pretty
image" />
    <figcaption>This is a beautiful
picture.</figcaption>
</figure>
```

But if your ebook files are XHTML (rather than HTML), the `<figure>` element is not allowed. By default, Sigil uses XHTML, so `<figure>` doesn't validate. (And, no, in Sigil you can't manually change your DOCTYPE (to `<!DOCTYPE html>`) because Sigil likes to clean your code, and changes it back.) When Sigil supports EPUB3 in future, this won't be an issue.

Links

A link is clickable text or a clickable image that, when clicked, takes the reader somewhere else. There are two kinds of links in an ebook:

- internal links, which point to other locations in the ebook;
- external links, which point to websites.

Internal links are most commonly used for cross-references and footnotes. External links for further reading (or watching and listening, as the case may be). External links will almost always open in a separate program, like a web browser or app. For instance, if you're reading an ebook on a smartphone and click a link to a YouTube video, the video will open either in the phone's default web browser, or in the YouTube app if it's installed.

In HTML, internal and external links look very similar. First, I'll explain how they work. Then I'll show you how to add them really easily in Sigil.

External links

Let's start with a basic external link:

```
 <a href="http://electricbookworks.com">This link
points to EBW's website.</a>
```

As you can see, a link consists of:

- an anchor tag `<a>`
- with one attribute, the `href`, or hyperlink reference.

The opening `<a>` and closing `</a>` tags enclose the text that will be clickable in the ebook, like this:

This link points to EBW's website.

Internal links

Let's create an internal link in this document to the 'Working with images' chapter. The HTML will look like this:

```
<a href="3-images.html">Click here to go to the
'Working with images' chapter.</a>
```

When rendered, this will look like this in the ebook:

Click here to go to the 'Working with images' chapter.

As you can see, jumping to the start of another HTML file is easy, you just use the file name as the `href` value, e.g. `href="filename.xhtml"`.

But what if we want to jump to a specific point in a chapter? For instance, to a specific element like a paragraph or heading or image? We have to mark that destination element somehow, so that we can point an `href` to it.

We do this by giving the element an ID. This is easy to do:

- make up an ID name (it must be unique in that HTML file),
- use no spaces, avoid capital letters, and don't start with a number, and
- include it as an attribute in the element's opening tag, like this: `<p id="bobsyouruncle">`

Then, when we make our link, we use that ID as the `href`, joined to the file name with a hash sign:

```
<a href="1-html.html#bobsyouruncle">Click here to
go to the specific paragraph I'm linking to.</a>
```

If you're linking to another place in the same HTML document, you don't even need the file name. You could just use a hash sign with the `id`:

```
<a href="#another-id">This goes to a point in this
same HTML doc.</a>
```

So link paths are relative to the file they're in.

Clickable images

It can be really useful to make an image clickable. To do this, you just wrap the `<a>` element around the `<img>` element, like this:

```
<a href="somewhere.xhtml">
        <img src="bob.jpg" alt="Uncle Bob" />
</a>
```

However, some ereaders, including some versions of Adobe Digital Editions, don’t support clickable images, which is annoying. So use them, but don’t rely on them to work 100 per cent, every time.

Adding links with Sigil

Sigil can create the link code for you. Whew!

- Highlight the text you want to make clickable.
- Go to `Insert > Link…`.
  - If you’re creating an internal link, pick a target from the list. Sigil lists all the HTML files and IDs in the book.
  - If you’re creating an external link, paste the full URL in the `Target` field.
- Click ‘OK’

If you’re linking internally, and the element you want to link to isn’t in the `Targets in the Book` list, you need to give that element an ID before linking to it. To do this:

- go to the element you’re linking to, and highlight it;
- go to `Insert > ID…`;
- make up an ID (no spaces, don’t start with a number, and avoid uppercase letters);
- click ‘OK’.

Now that element’s ID will be listed in the `Targets in the Book` list when you insert a link.

Metadata

Metadata is information about information. For instance, your driver's licence is metadata about you.

Epub ebooks must store some information about themselves so that machines know how to deal with them. You have to provide some of this metadata. Some metadata applies to the epub as a whole, some to specific files, and some even to specific elements inside files.

TECHNICAL NOTE

> Book-level and document-level metadata is actually stored in a special file called `content.opf`. Sigil will let you edit that file directly (just double-click it on the left), but editing it directly is not a good idea. You do not want to break something. Rather use Sigil's metadata editing tools.

Book-level metadata

Just to be a valid epub, epub-level metadata must include:

- an identifier (like an ISBN)
- the publisher's name
- a date of publication or creation.

It's also best practice to include:

- the title
- an author.

To add this metadata in Sigil, just go to `Tools > Metadata Editor`.

Document-level metadata

In your epub you'll have a bunch of HTML files, and each will likely correspond to a part of a book: cover, title page, copyright page, preface, and so on. If an ereader knows which HTML file represents each book part, it can do useful things. For instance, the first time you open an ebook, most good ereaders will jump you straight to the first chapter, skipping all the frontmatter. That's if it knows which HTML file is the first chapter.

Sigil lets us set this information as document-level metadata. Just right-click each HTML file and go to `Add Semantics…` then tick the book-part that corresponds to that chapter.

Only one file can correspond to each book part. And the most important semantic tag is the one Sigil calls 'Text'. That's the file that ereaders will jump to on first opening a new ebook.

In addition to adding semantics for HTML files, you *must* also add semantics for the cover image, usually `cover.jpg`, in your `Images` folder. Right-click the image file and select `Add Semantics > Cover Image`. This tells ereaders to use that image in its Library display, which is very important to users.

Element-level metadata

It's rare that you'd deliberately add metadata to elements within files, such as specific paragraphs or phrases, because you really don't have time or a clear need for it. But do keep in mind that it's possible, so that you can experiment with possibilities. For instance, imagine if you were creating a travel guide, and you marked up each phone number – for guest houses, museums, restaurants – like this:

```
<span itemprop="telephone">+27 21 671 1278</span>
```

Ereaders could use this tagging to recognise phone numbers and let you dial them from your smartphone with a single click.

If you're curious about the possibilities, read up online about 'microdata'.

Tables of Contents

In an ebook, the table of contents can take two forms:

- a 'Contents' page among the book's opening pages, as in a print book;
- a navigation list displayed by the ereading software (e.g. a menu in a slide-out panel or navigation screen).

It's optional whether to include a 'Contents' page. I generally avoid it, because it's extra work and an ereader's navigation menu is usually easier to use while reading. If you do create one, it's essentially a list of headings in the book, each hyperlinked to the relevant point in the book.

The navigation list, however, is mandatory. You store this list in a special file listing links to chapter or other headings in the book. Each ereader displays this table-of-contents list in a different way: for instance, as a navigation bar, a slide-out panel, a menu list, and so on.

At this point, the way we create the navigation list depends on whether we're creating epubs according to the EPUB2 or EPUB3 spec. Right now, Sigil only supports EPUB2. So we'll start there.

EPUB2 tables of contents

In EPUB2, you store the navigation list in a `toc.ncx` file (`toc` for Table of Contents, of course). You can see this file in Sigil, but *you do not want to edit it yourself.* The syntax is a real headache. Let Sigil code the `toc.ncx` file for you: use `Tools > Table Of Contents > Generate Table Of Contents…` to create and edit it.

Note that Sigil generates the TOC by collating all heading elements, from `<h1>` to `<h6>` . You can edit this manually in the TOC editor, but try to keep edits to a minimum. Ideally your heading levels should be so well-constructed that editing isn't necessary.

EPUB3 table of contents

In EPUB3 you store the navigation list in a dedicated XHTML file with any name you like. It's common to call it `nav.xhtml`, and to refer to it as the `toc nav`. You can then choose whether or not to include this XHTML file as content in your book – that is, as a 'Contents' page among the book's opening pages. (Technically speaking, the file is always listed in the OPF `manifest`, but only included in the `spine` if it must appear in the book content.)

We aren't going to go into detail about how to create that file here, since we're focusing on EPUB2 with Sigil. But you can read up about it online if you're curious.

TECHNICAL NOTE: EPUB3 WITH EPUB2 FALLBACK

> The EPUB3 specification allows for an epub to include a toc.ncx file as a fallback for ereaders that don't support EPUB3's EPUB Navigation Document. Hopefully, when Sigil becomes EPUB3 capable, it'll provide this fallback automatically.

Further reading: *EPUB 3 Best Practices*

If you want a great reference book, go get *EPUB 3 Best Practices* from O'Reilly Media.

Covers

There are many ways to create a cover in an epub ebook. You could just place an image in the first HTML file in the epub. But we generally get better-looking results with some very specific code.

Prepare the image

Get the front cover as a jpg image, name it cover.jpg, and add it to the `Images` folder of your epub.

To tell ereaders that it's the cover image, right-click it, select `Add Semantics`, and tick `Cover Image`. (Some ereaders won't display the cover in their library view unless you do this.)

Create the cover HTML

Next, add a new HTML document to your `Text` folder, and click-and-drag it so that it's the first HTML file in the list. Name the file 'cover.xhtml'.

Now, in the Code View of that file, replace the `<head>` and `<body>` elements with this code:

```
 <head>
        <title>Cover</title>
        <link rel="stylesheet" type="text/css"
href="../Styles/styles.css" />
</head\>

<body class="cover">
        <p class="cover">
                <img class="cover" alt="Cover"
src="../Images/cover.jpg" />
```

```
        </p>
</body>
```

Change the name of the CSS file here if yours isn't called `styles.css`.

Add the cover to CSS

Finally, add this code to your CSS file:

```
 \* Styles for cover.xhtml \*
body.cover {
        margin: 0;
        padding: 0;
        text-align: center;
}
p.cover {
        margin: 0;
        padding: 0;
        text-align: center;
}
img.cover {
        height: 100%;
        margin: 0;
        padding: 0;
}
```

That should do the trick on most good ereaders. Like every piece cover code I've found, it won't work perfectly everywhere.

You can find more technical detail about covers on the EBW Knowledge Base.

Testing and validation

Ebooks should be thoroughly checked, sometimes even proofread, just like print books. There are two parts to checking ebooks:

1. validating the code, and
2. checking the ebook on ereaders.

Validating the code

Sigil includes a useful validation tool that checks the technical correctness (validity) of your epub file. (It's called 'FlightCrew'.) Go to `Tools > Validate EPUB with FlightCrew` or hit the green tick button. If there are technical errors, Sigil will give you a list of them, and you can click on the items in the list to jump to the problem.

Checking on ereaders

Always, always test on a few different ereaders, and scroll through the whole book for a visual check. We recommend that you try to test on:

- a good reading app on a tablet and/or phone like iBooks, Google Play Books, Aldiko;
- a web-based reader like Google Play Books or Ibis Reader, or in Firefox using the EpubReader addon;
- Adobe Digital Editions on a computer (ADE has lots of problems, but it's still the biggest computer-screen ereader next to Kindle);
- an e-ink reader (e.g. Sony, COOL-ER, BeBook, etc.).

These tests should include a device with a small screen (less than, say, 5" diagonal; this can detect various formatting problems) and one with a large screen (above, say, 19"; this can detect low-res covers and missing page breaks).

A range of options like this will usually also include a device with an Adobe engine (i.e. one based on the Adobe Reader Mobile SDK, such as Sony

Readers and B&N Nook), and a device with a WebKit engine (e.g. Ibis Reader, iBooks).

On advanced tricks and bug fixing

In this course we rarely get to talk through advanced tricks and serious bug fixing. Those are things you're going to have to tackle when they arise. Things that'll come up at some stage:

- Mathematical notation: this is a real headache. usually, the only solution is to include images for every instance of maths.
- Sub-titles: note that `<h2>` can't be used as a sub-title to `<h1>`, because `<h2>` is a second-level of heading in the hierarchy, and signals the start of a new sub-section. Sub-titles usually have to be paragraphs with a special class, like `p.h1-subtitle`. Or just run them onto the `h1` with a colon, e.g. `Chapter 1: We set out early`.
- Transparency in images: gif and png images can have transparent areas. So if you have black line art, do you make your backgrounds white or transparent?
  - On sepia-background ereaders, images with white backgrounds look ugly, but
  - on white-text-on-black ereader settings (aka 'night mode'), black line art on a transparent background will be invisible.
- Embedding or linking to video: this is a real headache that you should read more about online.

Many other challenges await you. Always search and ask online, because someone out there has probably faced them before. And when you hit on a solution, be sure to share it to help others.

Your feedback

Training in a new, fast-changing field is hard, so we really need your honest feedback about this course. You can give very brief answers, though the more you can tell us the the better. You're welcome to answer anonymously.

1. Given your expectations before the course, did you learn what you wanted to learn?
   - Exactly what I wanted.
   - Mostly what I wanted.
   - Um. A bit of both.
   - Only a little of what I wanted.
   - Not at all what I had in mind.
2. What was the best thing about the course?
3. What about the course most needs to change?
4. Do you have suggestions for how to fix that?
5. We know these courses are expensive. For you, was the expense:
   - worth the investment,
   - too early to tell, or
   - not really worth it?
6. What did you think of the venue?
7. Assuming you had a positive experience, do you have any comments we could use publicly?

Feedback forms are a schlep, so thank you, really. Pop in at EBW for thank-you coffee any time.