Quick and dirty ePub tutorial (version 3)

Contents

Intro

Ok, you know nothing about HTML, CSS and ePub. You have no time or patience for standards and long tutorials. You just want to make your own ePub and you want it to look exactly like that amazing ebook you just downloaded from the MobileRead Library.

This is for you.

HTML in 1-2-3

HTML (Hyper Text Markup Language) is the starting point to make your ePub. HTML is the document file format in which web pages are written. Sounds complicated, but it isn't.

HTML documents are just text documents with tags to specify formatting.

How do you tell tags from normal text? Simple, tags are enclosed between "<" and ">" signs. For example, " " is a tag that marks the beginning of a new paragraph and " " is a tag that marks the end of a paragraph.

We are now ready to write our first HTML document. We'll use Notepad or any other simple text editor (but not Word or OpenOffice). Open your editor and type the following:

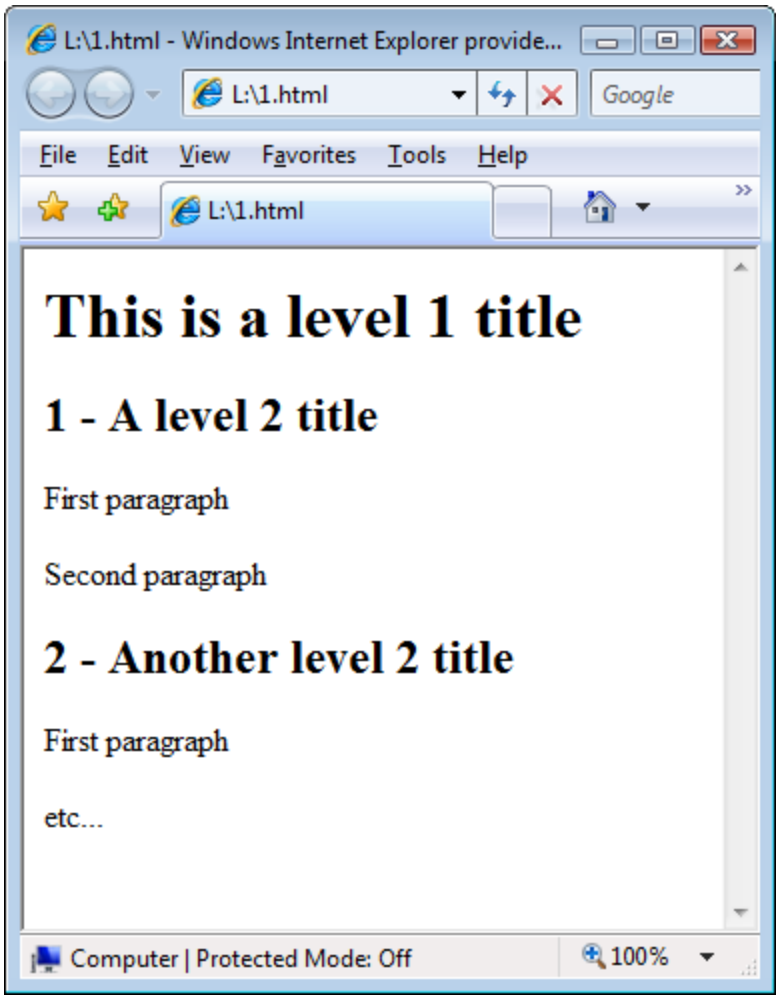
```
<h1>This is a level 1 title</h1>
<h2>1 - A level 2 title</h2>
<p>First paragraph</p>
<p>Second paragraph</p>
<h2>2 - Another level 2 title</h2>
<p>First paragraph</p>
<p>etc...</p>
```

What you see in red are the tags, the rest is the text. Tags that contain "/" are closing tags. Each piece of text is enclosed between opening and closing tags.

Tags <h1></h1> and <h2></h2> indicate level 1 and level 2 headers or titles and subtitles and tags indicate paragraphs. You can use up to 6 levels of headers (h1-h6).

Save this file as "1.html" (make sure it is not saved as "1.html.txt") and close your editor.

Now double click on "1.html". It should open in your internet browser. If you are using Internet Explorer, you will see something like this:



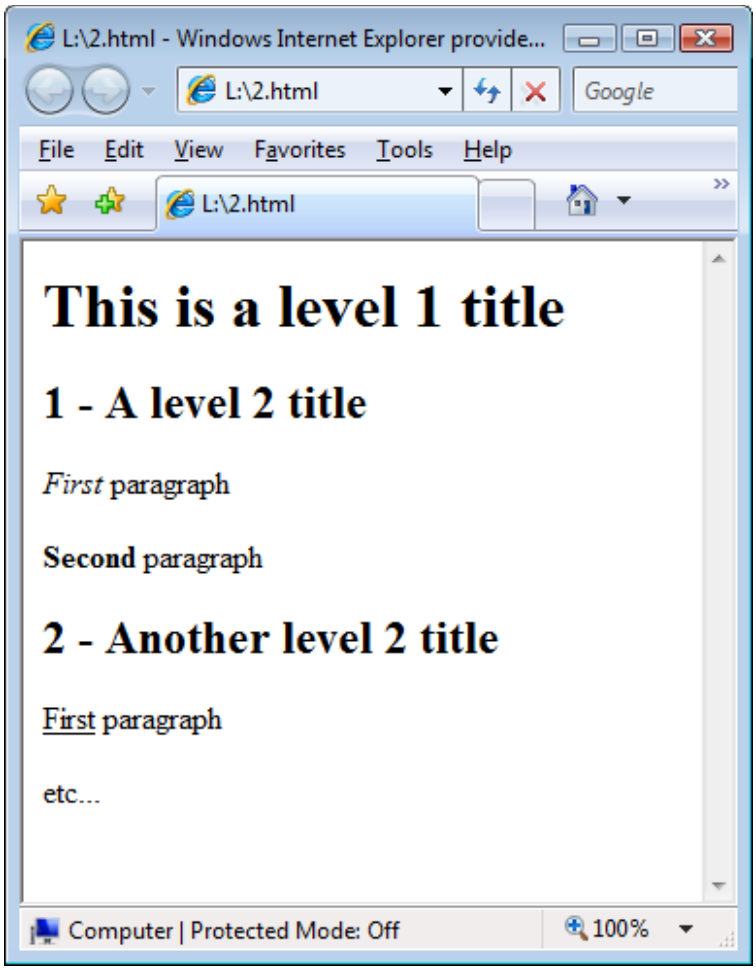
The styles for headers and paragraphs are the defaults of your browser. We'll see how to change them in a moment.

Now let's include some formatting. Copy your "1.html" file to "2.html" and open "2.html" with your text editor. Change it so that it looks like this:

```
<h1>This is a level 1 title</h1>
<h2>1 - A level 2 title</h2>
<p><i>First</i> paragraph</p>
<p><b>Second</b> paragraph</p>
<h2>2 - Another level 2 title</h2>
<p><u>First</u> paragraph</p>
<p>etc...</p>
```

Note: layout is not important here. You can add spaces, tabs and blank lines as desired to make your code more readable, it will make no difference in how it is displayed in the browser.

Save, close your editor, double click on "2.html" and you should see something like this:

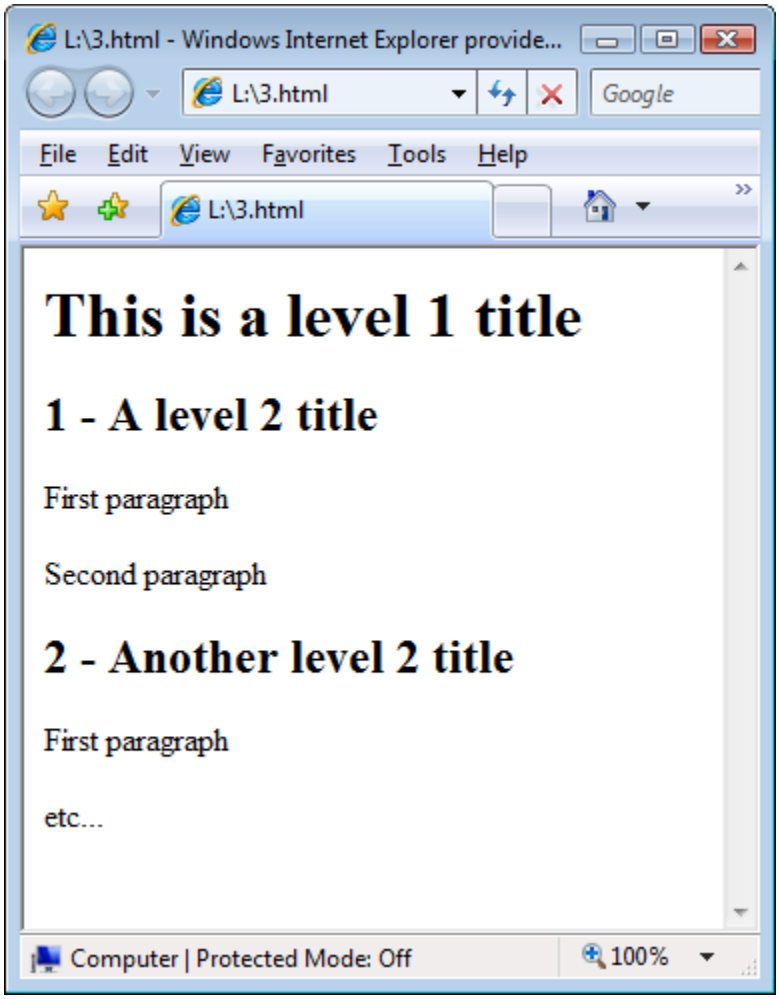


Tags format the inner text as *italics*, as **bold** and <u></u> as <u>underline</u>.

Now let's code our HTML file properly. Once again, make a copy of your "1.html" file, rename it as "3.html" and modify it as follows:

```
<html>
<head>
<meta name="Title" content="HTML micro tutorial" />
<meta name="Author" content="Pablo" />
</head>
<body>
<h1>This is a level 1 title</h1>
<h2>1 - A level 2 title</h2>
<p>First paragraph</p>
<p>Second paragraph</p>
<h2>2 - Another level 2 title</h2>
<p>First paragraph</p>
<p>etc...</p>
</body>
</html>
```

Save and open with your browser.



Ok, there's no difference with "1.html". That's because browsers are quite permissive and overlook HTML formatting issues when they can, but that will not always be the case, so let's start doing things properly from now on.

The contents of our previous "1.html" file are now enclosed between <body> and </body> tags. Not surprisingly, this is the body of the document. Before the body of the document we have a document header section enclosed between <head> and </head> tags. Inside the header we have two "metadata" lines, one for the document title and one for the document author. Finally, all the document is enclosed between <html> and </html> tags.

Of course this is not all there is to the HTML format, but it is enough for now. If you want to know more you can go to W3Schools.com's excellent HTML tutorial at:

http://www.w3schools.com/html/default.asp

CSS

Basics of CSS

Cascading Style Sheets (CSS) sounds even more complicated than HTML, but, of course, it is not.

CSS is just a method to tell the browser how each HTML element (paragraph, header, etc) should look.

One way to do this is to create a separate text file that specifies the format for each element and then include a reference to this file inside our HTML document.

So, open your text editor and type the following:

```
p {
        text-indent: 2em;
        text-align: justify;
        font-size: 1em;
        margin-top:0;
        margin-bottom:0;
}

h1 {
        font-size: 2.5em;
        text-decoration: underline;
        text-align: center;
        font-weight: bold;
        text-indent:0;
}

h2 {
        font-size: 2em;
        font-style: italic;
        font-weight: normal;
        text-align: left;
        text-indent:0;
}
```

Save as “style.css” (make sure it is not saved as “style.css.txt”).

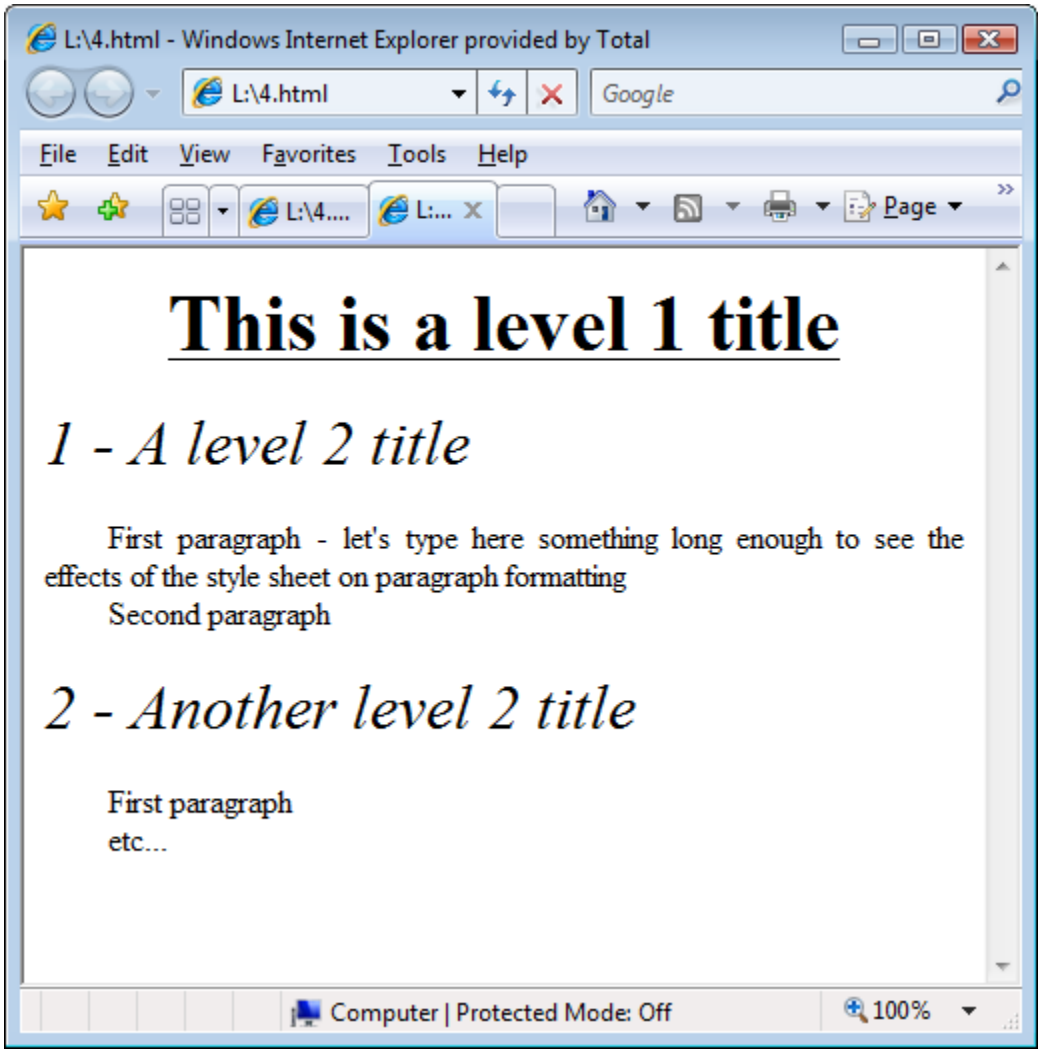
Now make a copy of “3.html” and rename as “4.html”, open with your text editor and modify as follows:

```
<html>
<head>
<meta name="Title" content="HTML micro tutorial" />
<meta name="Author" content="Pablo" />
<link rel="stylesheet" type="text/css" href="style.css" />
```

```
</head>
<body>
<h1>This is a level 1 title</h1>
<h2>1 - A level 2 title</h2>
<p>First paragraph - let's type here something long enough to see the effects of the style sheet on paragraph formatting</p>
<p>Second paragraph</p>
<h2>2 - Another level 2 title</h2>
<p>First paragraph</p>
<p>etc...</p>
</body>
</html>
```

The new line links the HTML document with the "style sheet" we have just created. Make sure both "4.html" and "style.css" are in the same folder.

Double click "4.html" to open it in your browser. You should see something like this:



Note that:

- The level 1 header is in bold, centered and underlined
- Level 2 headers are in italics
- Paragraphs are fully justified and indented, with no separation between them.

Now look at your "style.css" file and try to relate each line to what you see in your browser. They should be self-explanatory, except the "em" unit. To understand it, think of "1em" as 100%. So h1 font size is 250% and h2 is 200% (in relation to the paragraph font size, set at 100%).

So why go to all this trouble when you could have achieved the same result by inserting appropriate HTML tags inside the document?

There are several reasons:

1. A format change in one element in the CSS applies to all elements of that type in the document (no need to replace on a one-by-one basis).
2. The resulting document looks cleaner and easier to understand.
3. You can reuse your CSS for other documents.
4. Switching between different CSS files you can rapidly change the look of your document completely.

The "class" selector

The next step is defining class selectors inside the CSS. Let's show it with an example:

Open "style.css" file with your text editor and add the following lines at the end:

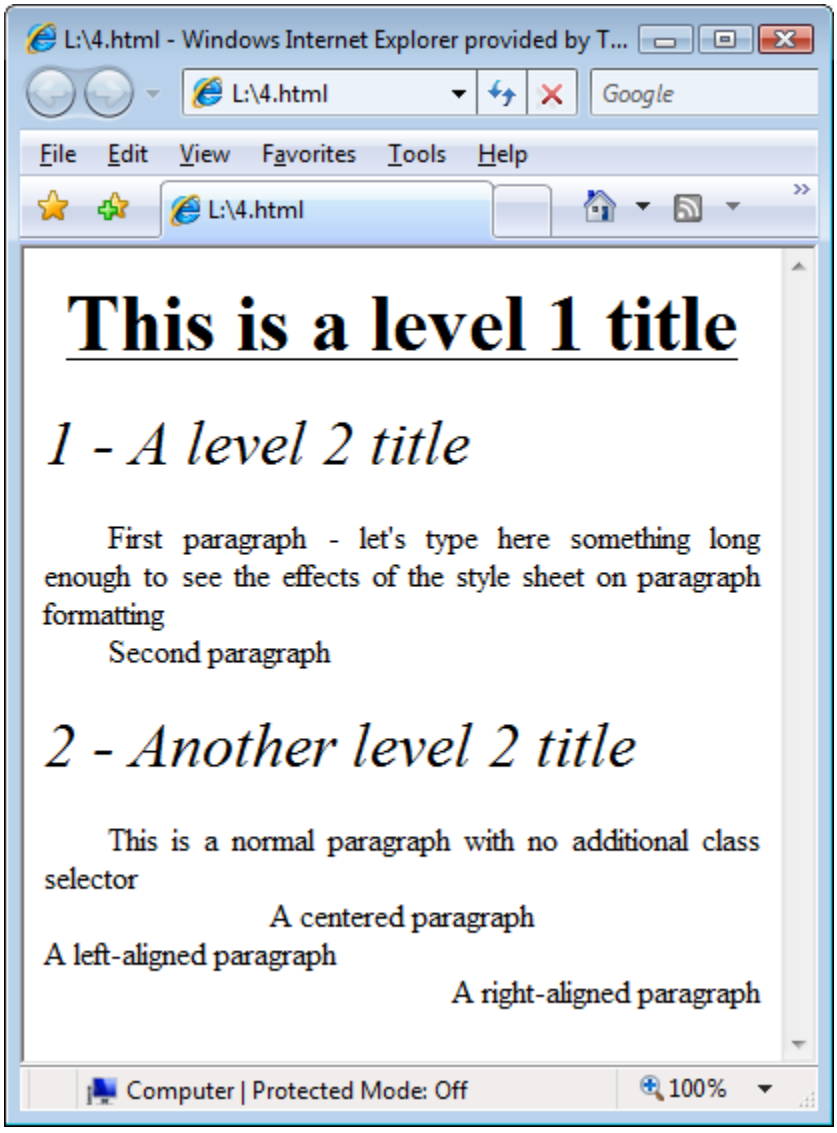
```
.center {text-align: center; text-indent: 0;}
.left {text-align: left; text-indent: 0;}
.right {text-align: right; text-indent: 0;}
```

Save and close.

Now open your "4.html" file and modify the part between <body> and </body> as follows:

```
<h1>This is a level 1 title</h1>
<h2>1 - A level 2 title</h2>
<p>First paragraph - let's type here something long enough to see the effects of the style sheet on paragraph formatting</p>
<p>Second paragraph</p>
<h2>2 - Another level 2 title</h2>
<p>This is a normal paragraph with no  additional class selector</p>
<p class="center">A centered paragraph</p>
<p class="left">A left-aligned paragraph</p>
<p class="right">A right-aligned paragraph</p>
```

Save and open in your browser. You should see something like this:



Note that:

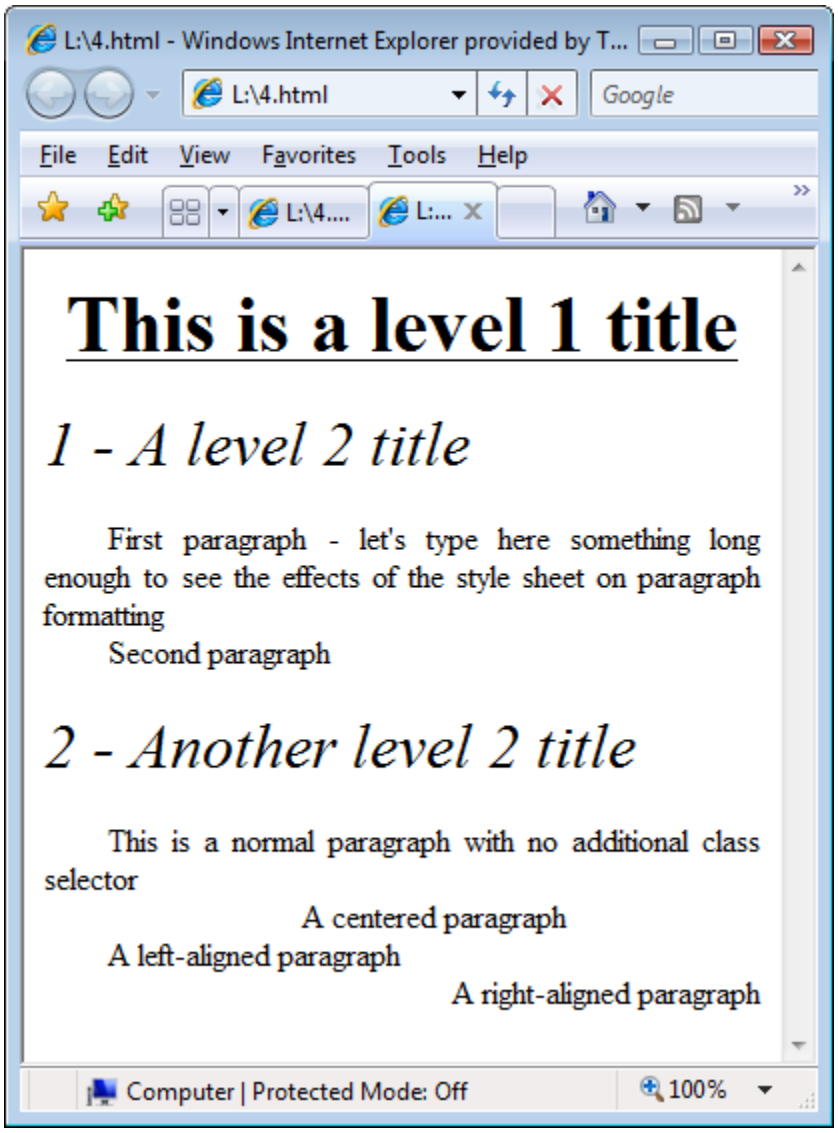
- The first paragraph under the second level 2 title is indented and fully justified, as defined in the "p" entry in the CSS.
- The other paragraphs are formatted according to their selectors. The properties defined in the selectors override the properties defined for "p".

We could also have done all this in another way:

- Remove the "text-align" property from the definition of "p" in the CSS
- Create a new ".justify" selector with the value of "text-align" set at "justify"
- Code normal paragraphs in the document with <p class="justify">

But if most paragraph will be fully justified, it is simpler to define with “text-align: justify” and code all normal paragraphs like “ Paragraph ”.

You may wonder why we had to include “text-indent: 0;” in the selector definitions. If you remove them and reopen “4.html”, you will note the difference in the centered and left-aligned paragraphs.



We can also apply these selectors to other HTML tags, for example, headers.

Once again, this is not all there is to CSS. To know more, go to

http://www.w3schools.com/css/css_intro.asp

And now what?

A full presentation of HTML and CSS are outside the scope of this document. The tutorials at W3Schools are probably the best available in the web, but it is of little use to make an in-depth study at this stage. You now know enough to start playing with the Epub format.

The ePub format

Basics of the ePub format

What you need to know about an ePub document:

1. An ePub document is a compressed file. You can look inside with your favorite compression program. You may need to rename it from "ebook.epub" to "ebook.zip" to do this. What you will find is something like this:

   (blue for directories and green for files)

   - Ebook.epub
     - META-INF
       - container.xml
     - OEBPS
       - Fonts
       - Images
       - Styles
       - Text
       - content.opf
       - toc.ncx
     - mimetype

   As we will not be building our ePubs manually, we are not very interested in what each file and folder is for (Sigil will generate and fill this structure for us). Some of them are quite obvious, though:
   - The Fonts folder contains font files if font-embedding is used in the document
   - The Images folder contains all images used in the document
   - The Styles folder contains one or more CSS stylesheets
   - The Text folder contains one or more (x)html files with the text of the book
2. The contents of the book are stored in one or more ".html" or ".xhtml" files in the Text folder. You don't need to worry about this new "xhtml" term. You will be able to understand xhtml files with what you already know about html. It is a good practice to split the book into several files, for example, one file for the cover and one for each chapter, though it is not mandatory.

Some ebook readers put restrictions on the maximum size of these files and are not able to open them when this size is exceeded.

3. You can use one or more CSS stylesheets.
4. You can create a *table of contents* for your book to be accessed with the appropriate tool of your device or reading application. It is not necessary to create a TOC inside the book itself (sometimes known as an *inline TOC*), but some people like to do so.
5. You can store *metadata* in your ePub. Examples of metadata are: Title, Author, Book Producer, Description, Publisher, Subject, etc.
6. You can put links in your document to ease navigation.
7. You can place images anywhere in your document.
8. Headers and footers are not supported.
9. You cannot code real footnotes. Normal practice is to put all footnotes at the end of the book (end notes) and link from body to note and back.
10. Each ebook reader or reading application has its own set of fonts for displaying books, so the look of a book will vary from device to device. If you want to force one or more specific fonts you have to *embed* them in the book and modify your CSS accordingly. Not all devices or reading applications support font embedding, though. In readers that support user selectable fonts, font embedding will block this feature.

Sigil

Sigil, created by Strahinja Markovic and currently maintained by John Schember is probably one of the greatest pieces of software of all time, even though it is still in the development stage and, in addition, it is free, open source and multiplatform.

Sigil currently imports HTML, TXT and ePub files. The output is always ePub.

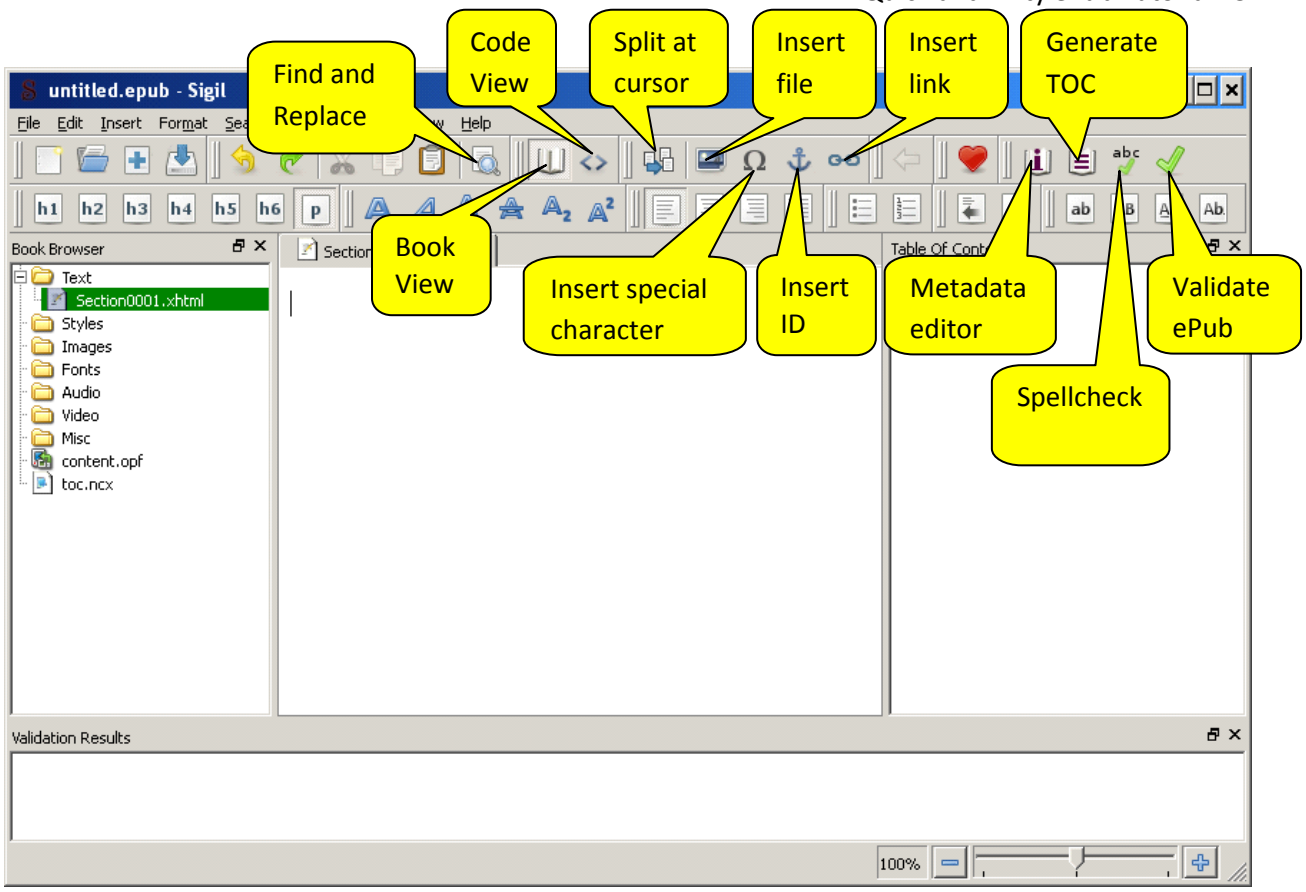
So go find the latest stable version in the web (http://code.google.com/p/sigil), download and install it. In this tutorial we will use version 0.7.1.

For a full description Sigil's features see the online user guide at

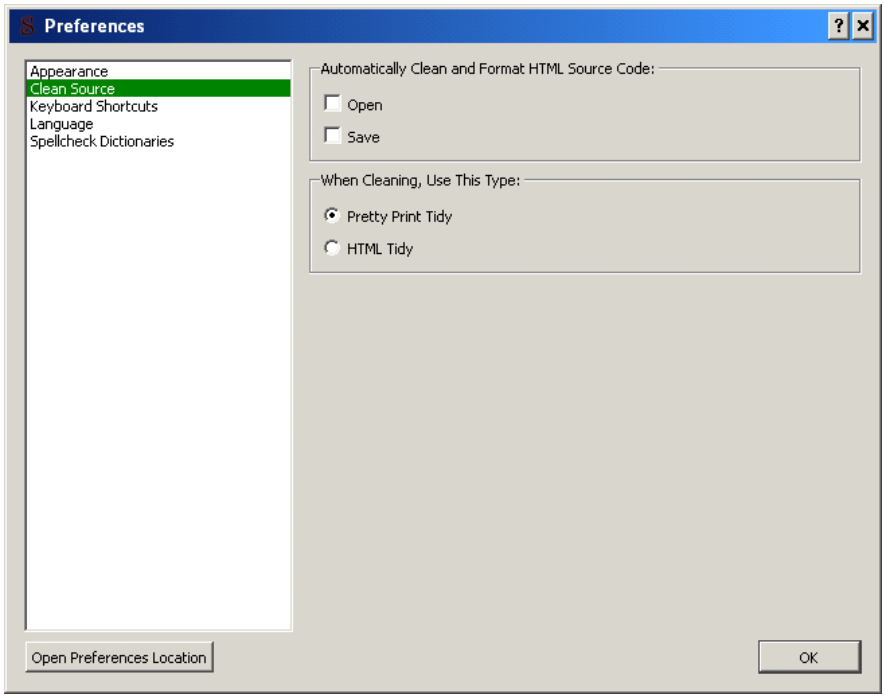
http://web.sigil.googlecode.com/git/files/OEBPS/Text/introduction.html

Running Sigil for the first time

The first time you run Sigil (without loading any documents), you will see the following screen, in which the most important functions have been highlighted.



Before starting work on our books, we should go to **Edit→Preferences→Clean Source** (see below)



Here we have to decide when and how much automatic cleaning and tidying we want Sigil to perform on our *HTML source code*.

If no automatic cleaning is desired we have to leave the “open” and “save” checkboxes unchecked. If we choose to allow automatic cleaning when the document is opened or when it is saved (or both), we have two possibilities:

Pretty Print Tidy: the *HTML source code* is formatted to make it easier to read and basic error correction is performed, like adding missing closing tags.

HTML Tidy: the same as Pretty Print Tidy plus converting some of your tags into internal styles. HTML Tidy is not perfect and sometimes the cleaning goes too far, causing (unwanted) loss of text.

Some people (myself included) find the “HTML Tidy” function annoying, since it tends to change perfectly valid HTML code unnecessarily. Sometimes the results are less “clean” than the original. Of course, this is a matter of opinion, but for beginners it is best to avoid this option.

We’ll select “Open” and “Pretty Print Tidy” to start with.

Our working file

We will use a longer html file to load into Sigil. Save this file as “5.html”.

```
<html>
<head>
<meta name="Title" content="Book Title" />
<meta name="Author" content="Book Author" />
<link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
<h1>Book Title</h1>
<h1>Book Author</h1>
<h2>Part 1</h2>
<h3>Chapter 1</h3>
<p class="first">Opening paragraph of Chapter 1 of Part 1. This paragraph should
have no indentation</p>
<p>Second paragraph. This paragraph should be indented to separate
it from the previous paragraph. As an alternative, we could put a line
between paragraphs, but that would waste a lot of space in the reader's
screen.</p>
<p>Last paragraph of this thread, a new section follows.</p>
<p class="scenechange">* * *</p>
<p class="first">Opening paragraph of a new section inside the same chapter. This
paragraph needs no indentation.</p>
<p>Etc., etc.</p>
<h3>Chapter 2</h3>
<p class="first">Opening paragraph of Chapter 2 of Part 1. This paragraph should
```

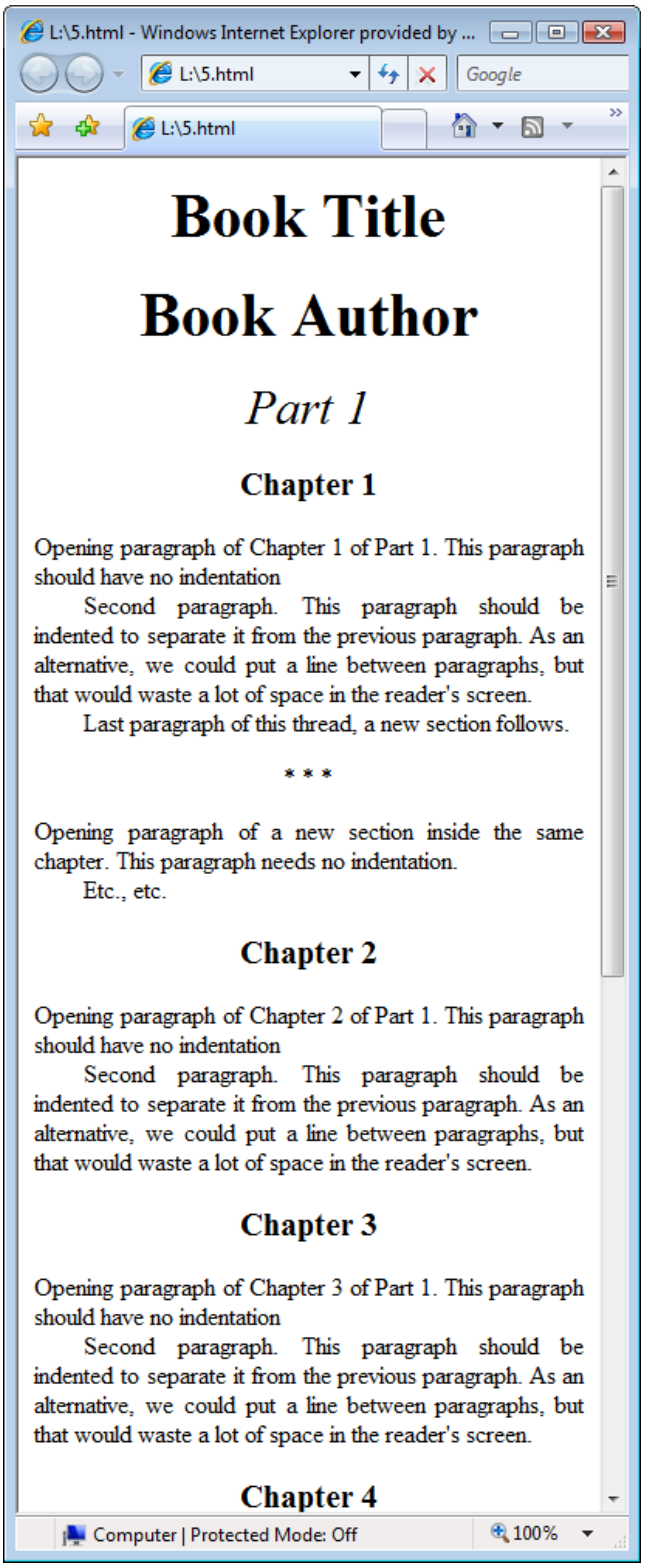
```
have no indentation</p>
<p>Second paragraph. This paragraph should be indented to separate
it from the previous paragraph. As an alternative, we could put a line
between paragraphs, but that would waste a lot of space in the reader's
screen.</p>
<h3>Chapter 3</h3>
<p class="first">Opening paragraph of Chapter 3 of Part 1. This paragraph should
have no indentation</p>
<p>Second paragraph. This paragraph should be indented to separate
it from the previous paragraph. As an alternative, we could put a line
between paragraphs, but that would waste a lot of space in the reader's
screen.</p>
<h3>Chapter 4</h3>
<p class="first">Opening paragraph of Chapter 4 of Part 1. This paragraph should
have no indentation</p>
<p>Second paragraph. This paragraph should be indented to separate
it from the previous paragraph. As an alternative, we could put a line
between paragraphs, but that would waste a lot of space in the reader's
screen.</p>
<h2>Part 2</h2>
<h3>Chapter 1</h3>
<p class="first">Opening paragraph of Chapter 1 of Part 2. This paragraph should
have no indentation</p>
<p>Second paragraph. This paragraph should be indented to separate
it from the previous paragraph. As an alternative, we could put a line
between paragraphs, but that would waste a lot of space in the reader's screen.</p>
<h3>Chapter 2</h3>
<p class="first">Opening paragraph of Chapter 2 of Part 2. This paragraph should
have no indentation</p>
<p>Second paragraph. This paragraph should be indented to separate
it from the previous paragraph. As an alternative, we could put a line
between paragraphs, but that would waste a lot of space in the reader's
screen.</p>
</body>
</html>
```

Our CSS

```
p {
	text-indent: 2em;
	text-align: justify;
	font-size: 1em;
	margin-top:0;
	margin-bottom:0;
}
h1 {
	font-size: 2.5em;
	text-align: center;
	font-weight: bold;
	text-indent:0;
}
h2 {
	font-size: 2em;
	text-align: center;
	font-style: italic;
	font-weight: normal;
}
h3 {
	font-size: 1.3em;
	font-weight: bold;
	text-align: center;
}
h4 {
	text-align: center;
}
.center { text-align: center;}
.first {text-indent: 0;}
.scenechange {text-indent: 0; text-align: center; margin-top:1em; margin-bottom: 1em;}
```

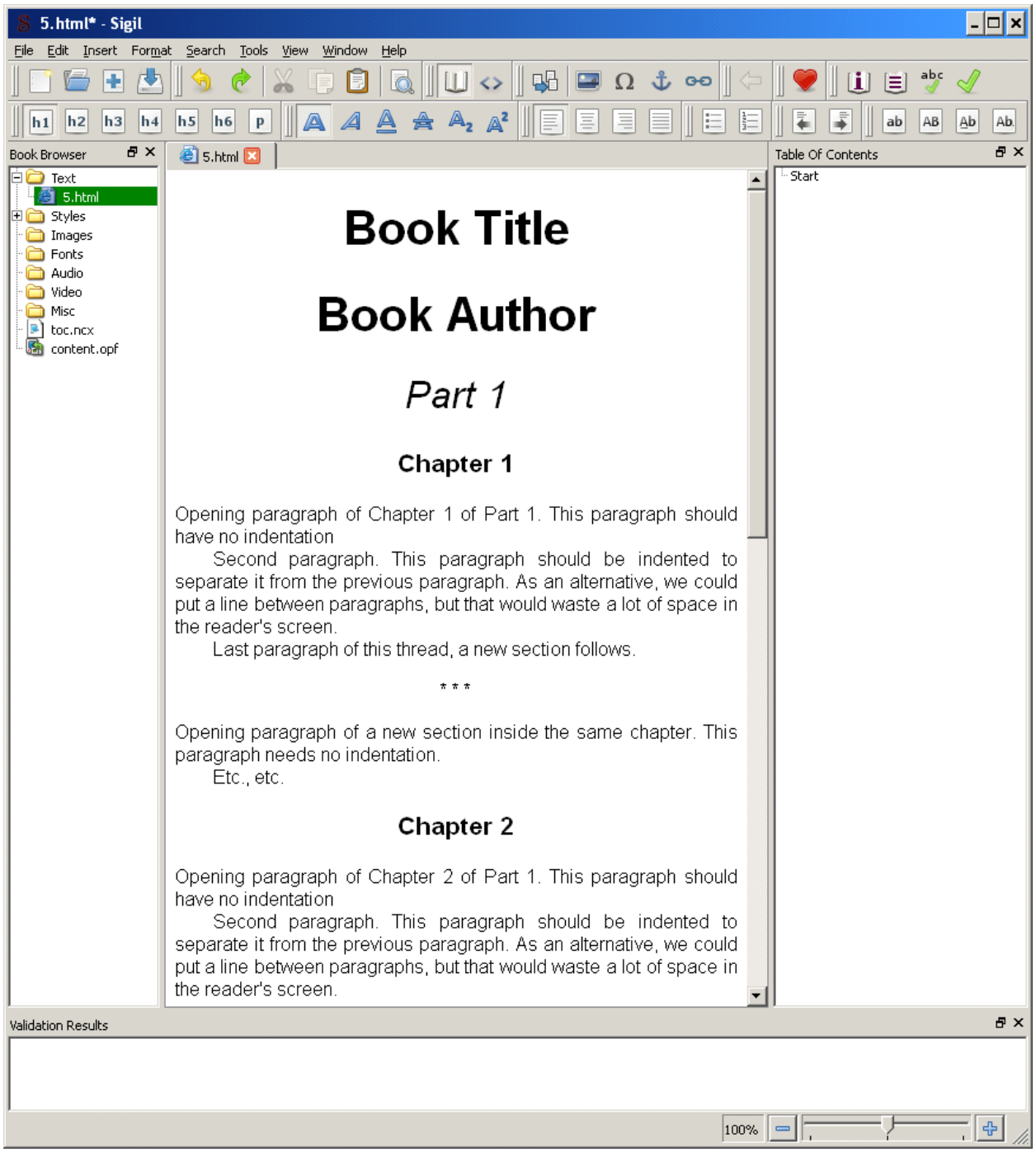
Save this file as “style.css”.

This is how it looks in Internet Explorer:



Importing with Sigil

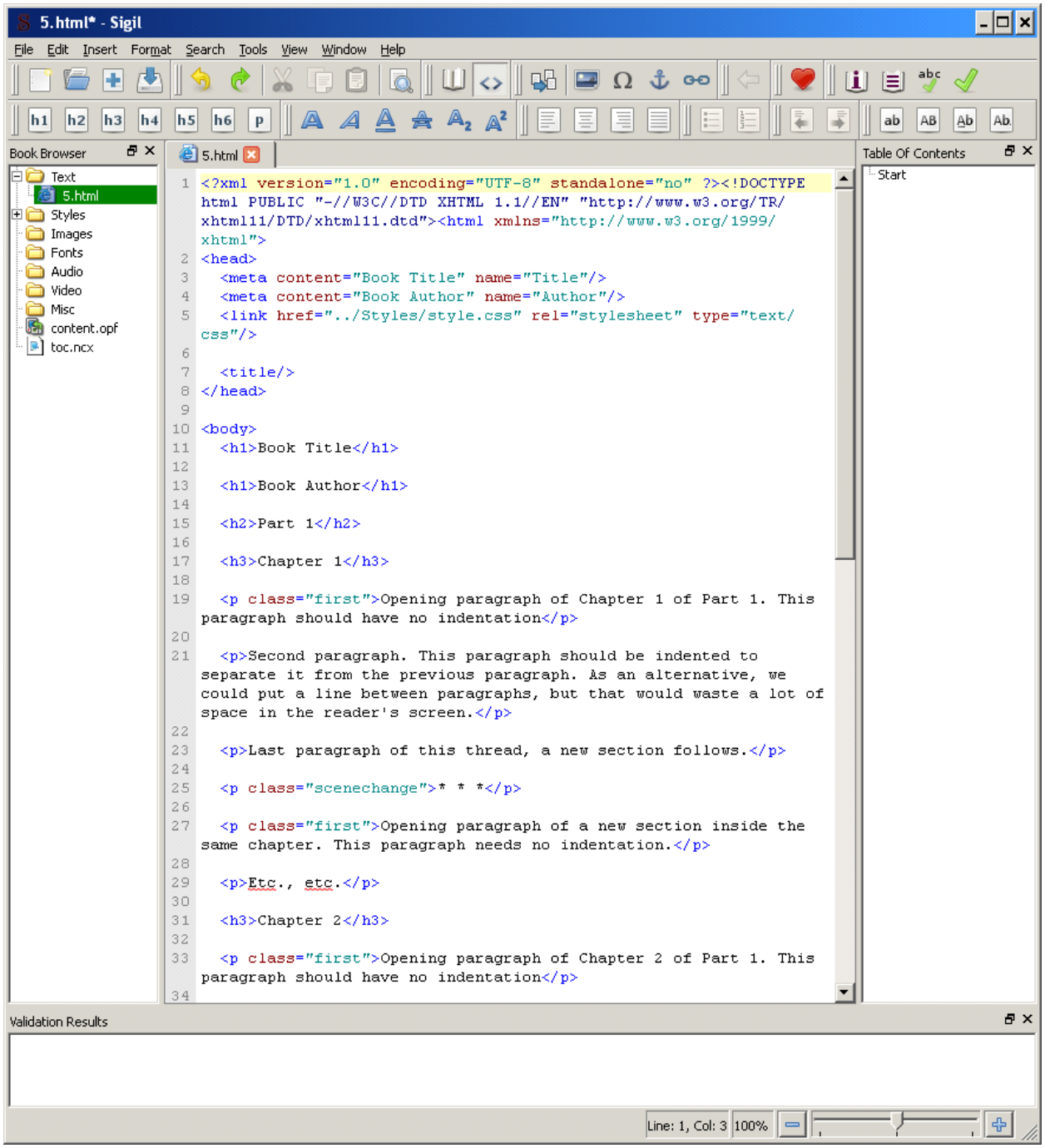
Now let's open this file with Sigil (**File→Open**). We should see something like this:



If you see something different, click on the “Book View” icon.

The view in the center is called “Book view”. It is similar to what we see in an Internet browser. This view is useful to make minor corrections to the text (typos, OCR scan errors, punctuation, etc.), but doing more than that (creating new paragraphs or headings, inserting blank lines, changing formatting) is not a good idea (more on this later).

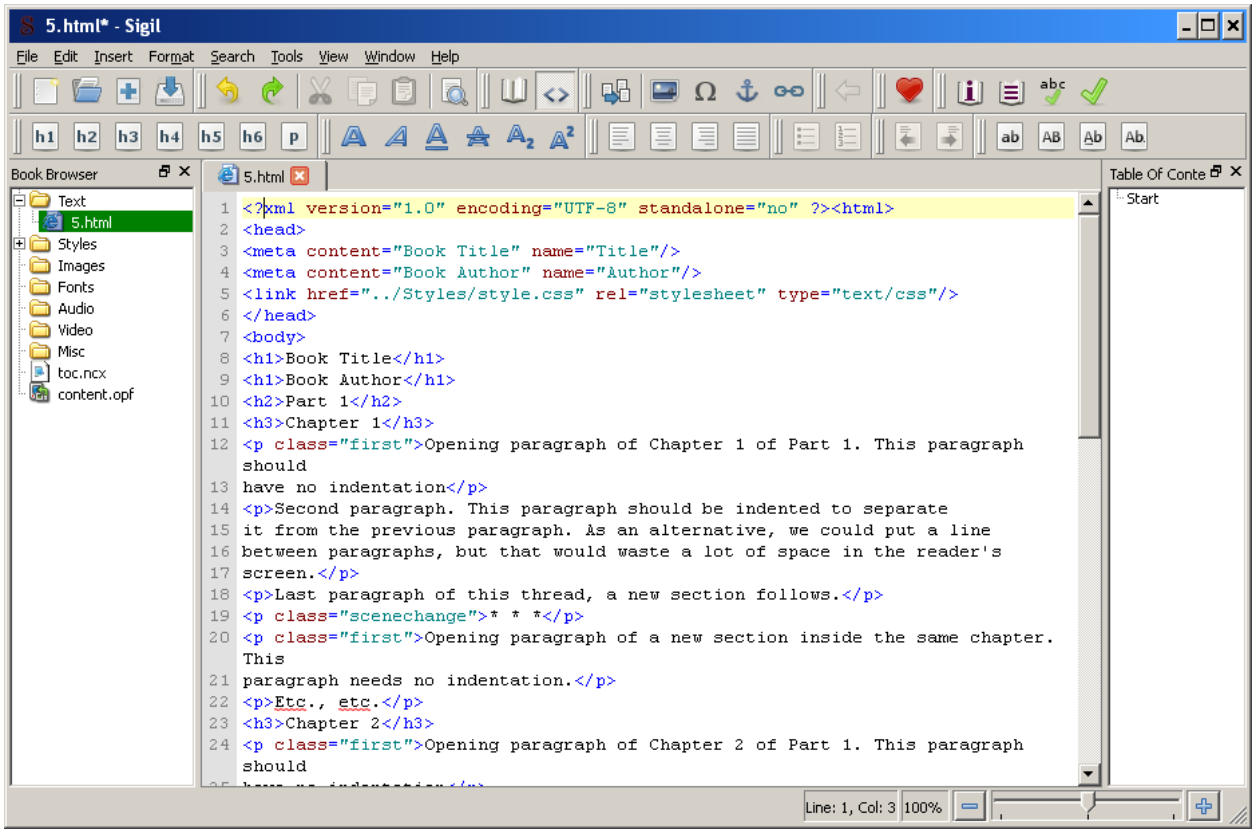
Clicking on the “Code View” icon or selecting **View→Code view** in the menu bar, we get:




```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?><!DOCTYPE
html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/
xhtml11/DTD/xhtml11.dtd"><html xmlns="http://www.w3.org/1999/
xhtml">
<head>
  <meta content="Book Title" name="Title"/>
  <meta content="Book Author" name="Author"/>
  <link href="../Styles/style.css" rel="stylesheet" type="text/
css"/>

  <title/>
</head>

<body>
  <h1>Book Title</h1>

  <h1>Book Author</h1>

  <h2>Part 1</h2>

  <h3>Chapter 1</h3>

  <p class="first">Opening paragraph of Chapter 1 of Part 1. This
paragraph should have no indentation</p>

  <p>Second paragraph. This paragraph should be indented to
separate it from the previous paragraph. As an alternative, we
could put a line between paragraphs, but that would waste a lot of
space in the reader's screen.</p>

  <p>Last paragraph of this thread, a new section follows.</p>

  <p class="scenechange">* * *</p>

  <p class="first">Opening paragraph of a new section inside the
same chapter. This paragraph needs no indentation.</p>

  <p>Etc., etc.</p>

  <h3>Chapter 2</h3>

  <p class="first">Opening paragraph of Chapter 2 of Part 1. This
paragraph should have no indentation</p>
```

This view is called “Code view”. This is our original html code, slightly modified by Sigil. This is where we’ll do most of our editing. Notice that color-marking make the code highly readable in comparison with Notepad.

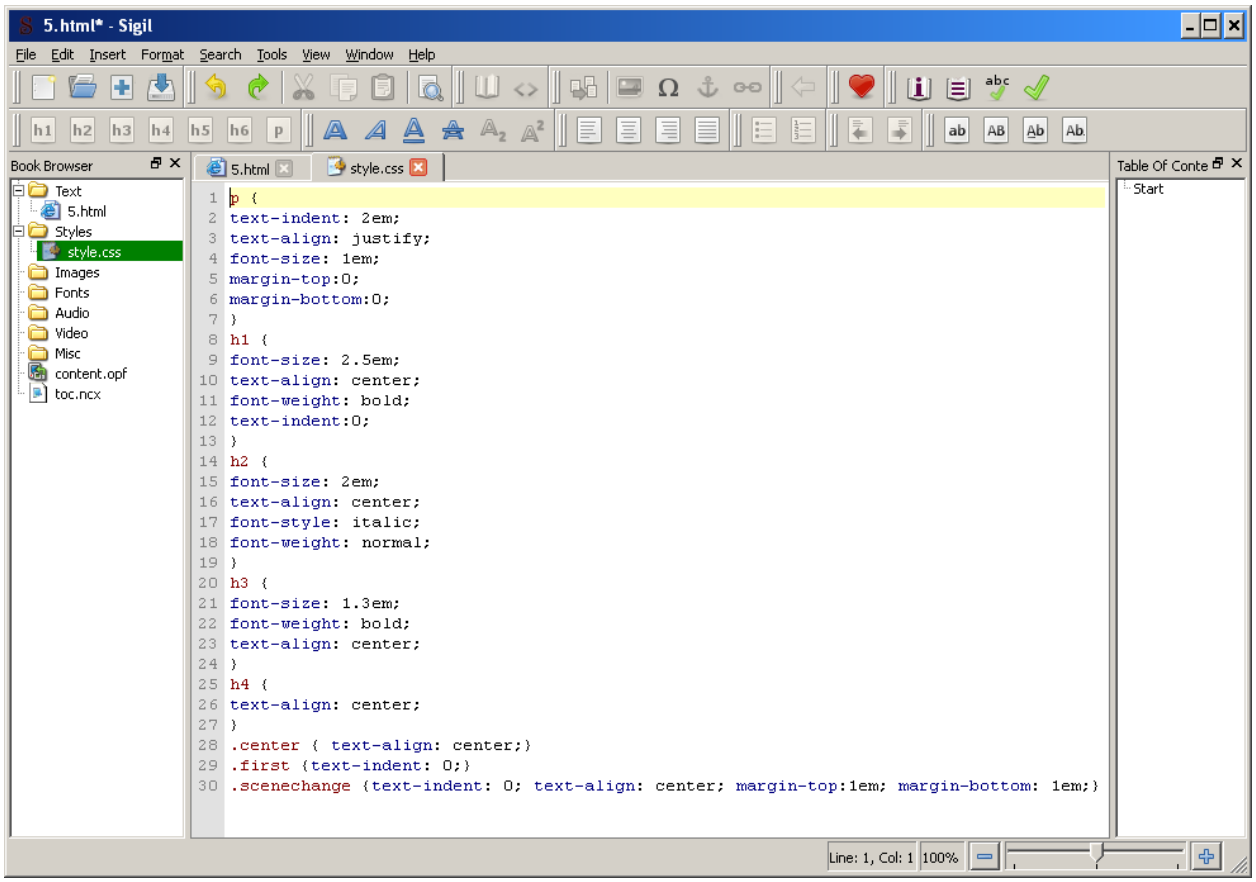
Before proceeding, change your preferences to disable automatic cleaning (uncheck the “open” checkbox) and reopen the file (Sigil has no “close” option, just open the same file again and “discard” changes when prompted). This is how it looks (in “Code View”) with no cleaning:



As you see, enabling Pretty Print Tidy cleaning provides a clearer view of the HTML source code. Now change to “Book View”. You will notice that enabling or disabling cleaning makes no difference in this view, which is not surprising.

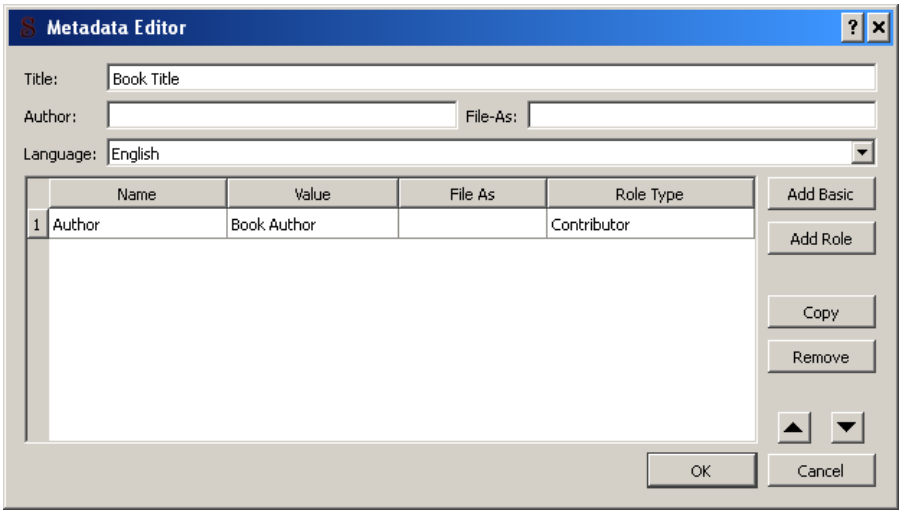
Now re-enable cleaning and reopen the file.

Expanding the “Styles” folder in the “Book Browser” panel on the left, we’ll find our “style.css” stylesheet. Double clicking on it we get:



As we see, the CSS has been imported with no changes.

Clicking on the Metadata Editor icon, we get this dialog:



The title and author have been imported from the header section of our html file, the language is the default set in **Edit→Preferences→Language→Default Language for Metadata**. Note that the "Author"

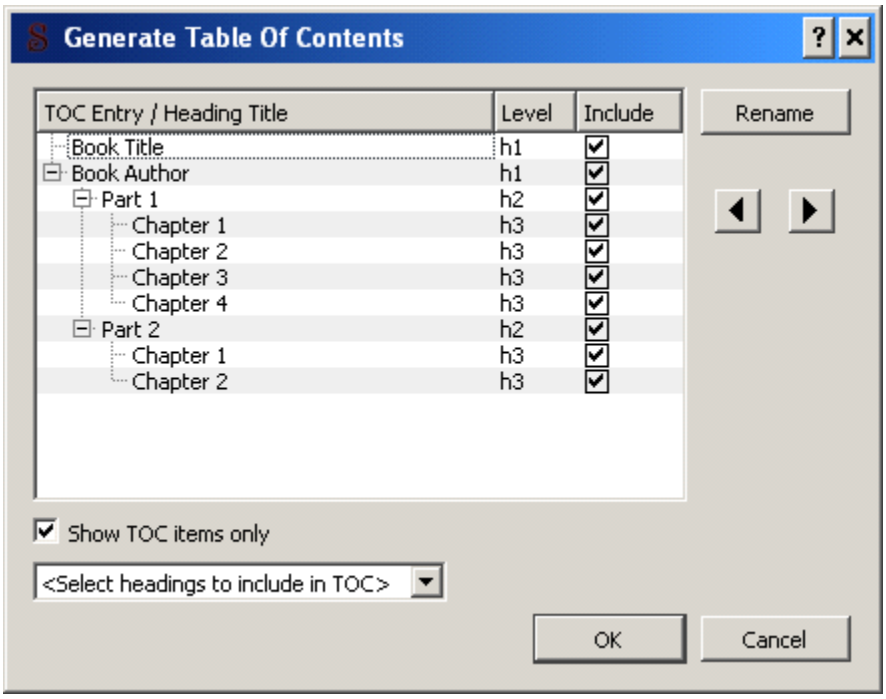
attribute shows in the grid but not in the “Author” and “File-As” textboxes. In the “Author” textbox we should type the author’s name as “FirstName LastName” and in the “File-As” textbox as “LastName FirstName”. After that we can remove the “Author” field in the grid.

The “Add Basic” and “Add Role” buttons allow us to include additional metadata fields and information, like “subject”, “description”, “editor” and so on. All the information included in this way is stored in the “metadata” section of the “content.opf” file mentioned before.

The Metadata Editor should be opened at least once and exited pressing the OK button, so that the “Language” property is stored in the opf file, otherwise, we’ll get an ePub validation error later.

Creating a table of contents

Clicking on the “Generate TOC” icon, we get the Table of Contents editor dialog:



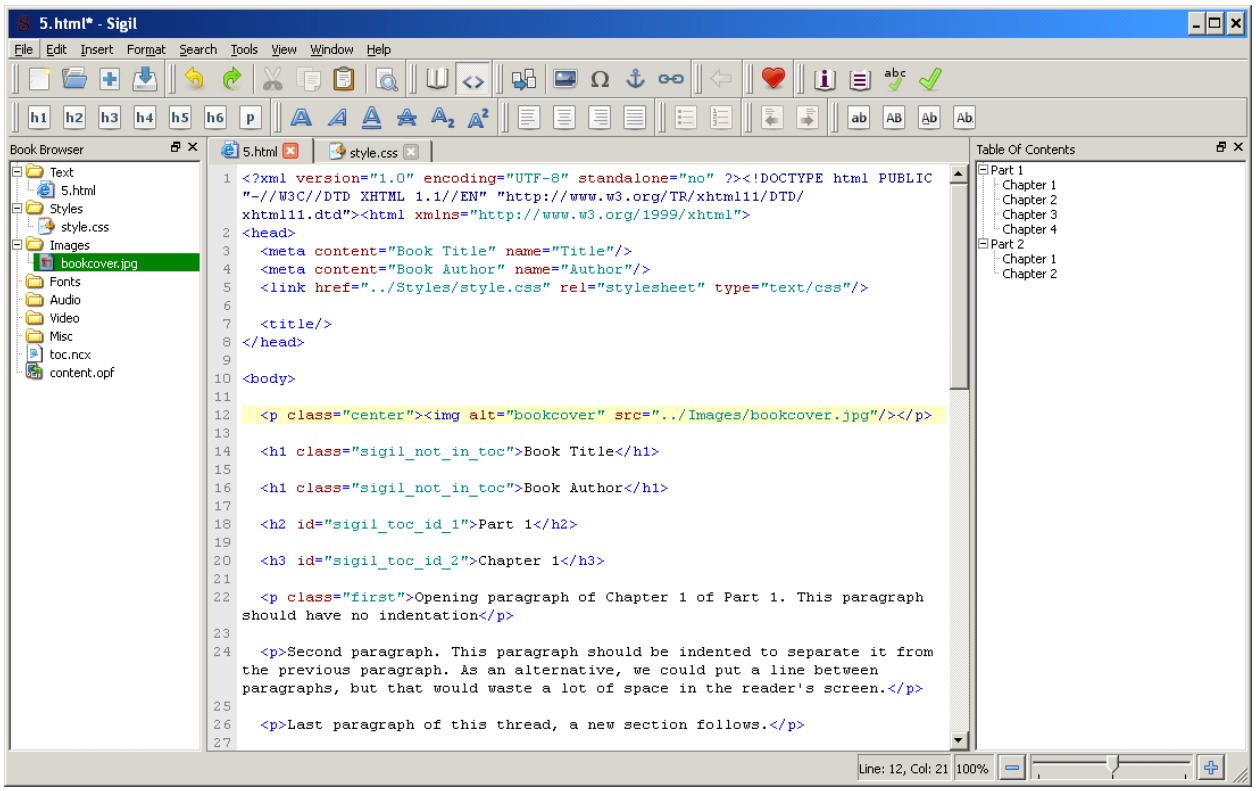
Sigil builds the multilevel TOC using the headings. We have used <h1> headings for Title and Author, <h2> for “parts” and <h3> for chapters. As the Title and Author are not part of the TOC, we have to exclude them, unchecking the boxes at their right. Excluded items disappear from the dialog. To see them again (and eventually re-include them in the TOC), just uncheck the “Show TOC items only” checkbox. The resulting TOC is stored in the “navmap” section of the “toc.ncx” file mentioned before.

Adding a cover image

You can add a cover image if you like. Supported formats are “jpg”, “gif” and “png”. Image size should be around 590 x 750 pixels, which is suitable for most ebook readers.

To add the image we must first include it in the “Images” folder in the “Book Browser” panel to the left. To do that, right-click on “Images” and select “Add existing files…”, then browse to your image.

In Code View, place the cursor below the <body> element, click on the “Insert File” icon and select your cover image. A new “img” element pointing to the image will be added. Enclose this element between and tags, as shown in the following screenshot:



You can then change to Book View to check that your image has been correctly placed.

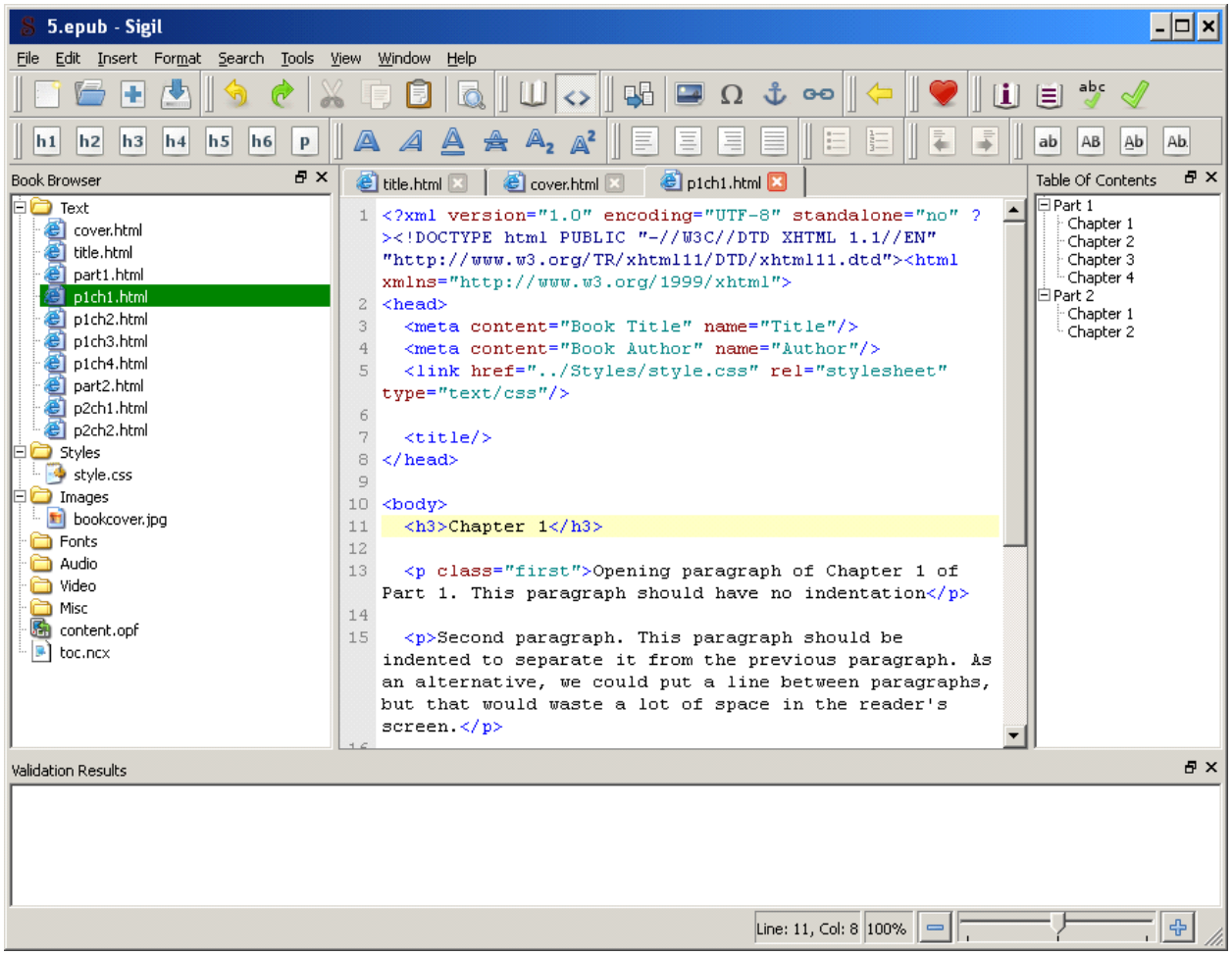
This is a good time to save our file as ePub, selecting **File→Save As** in the menu bar. All further editing will be done on this file.

Splitting text

As mentioned before, some readers put restrictions on the maximum size of text files, so it is necessary to split the text in several files, for example:

- One file for the cover image
- One file for the Book Title/Book Author page
- One file for each chapter
- One file for the notes

To do this, change to Code View, locate the blank line before the place where the file is to be split and press the “Split at cursor” icon. When finished you should see something like this:



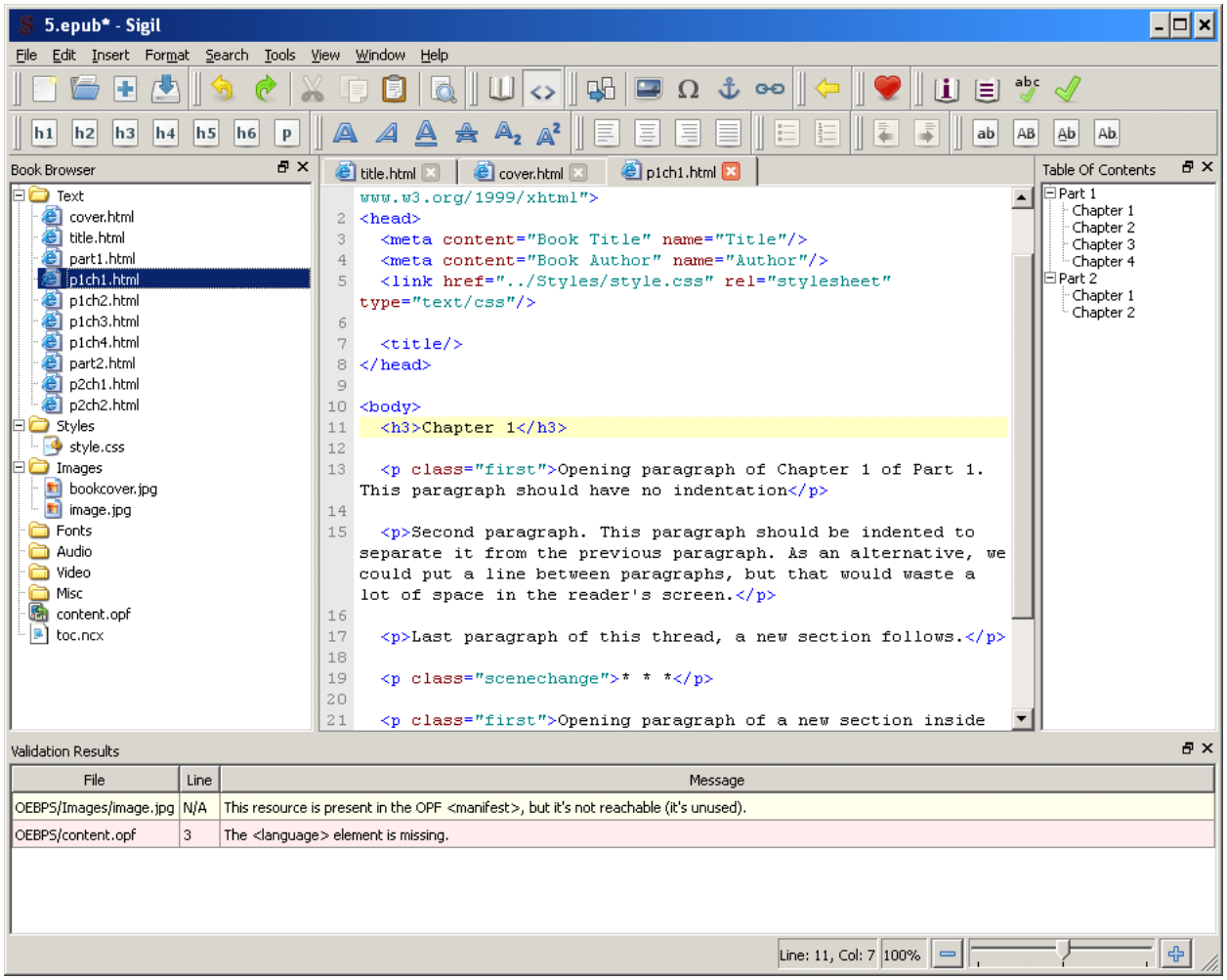
Each new file is labeled as “SectionXXXX.html” (XXXX = 0000 to 9999). The files can be renamed by right-clicking on the file name in the Book Browser panel and selecting “rename” in the context menu. In the example the files have been renamed to give a better idea of the contents of each one.

To explore each file, just double-click on its name in the “Book Browser” panel. Each file opens in a new tab.

After splitting files, it is a good idea to regenerate the TOC using the “Generate TOC” icon..

Validating your ePub

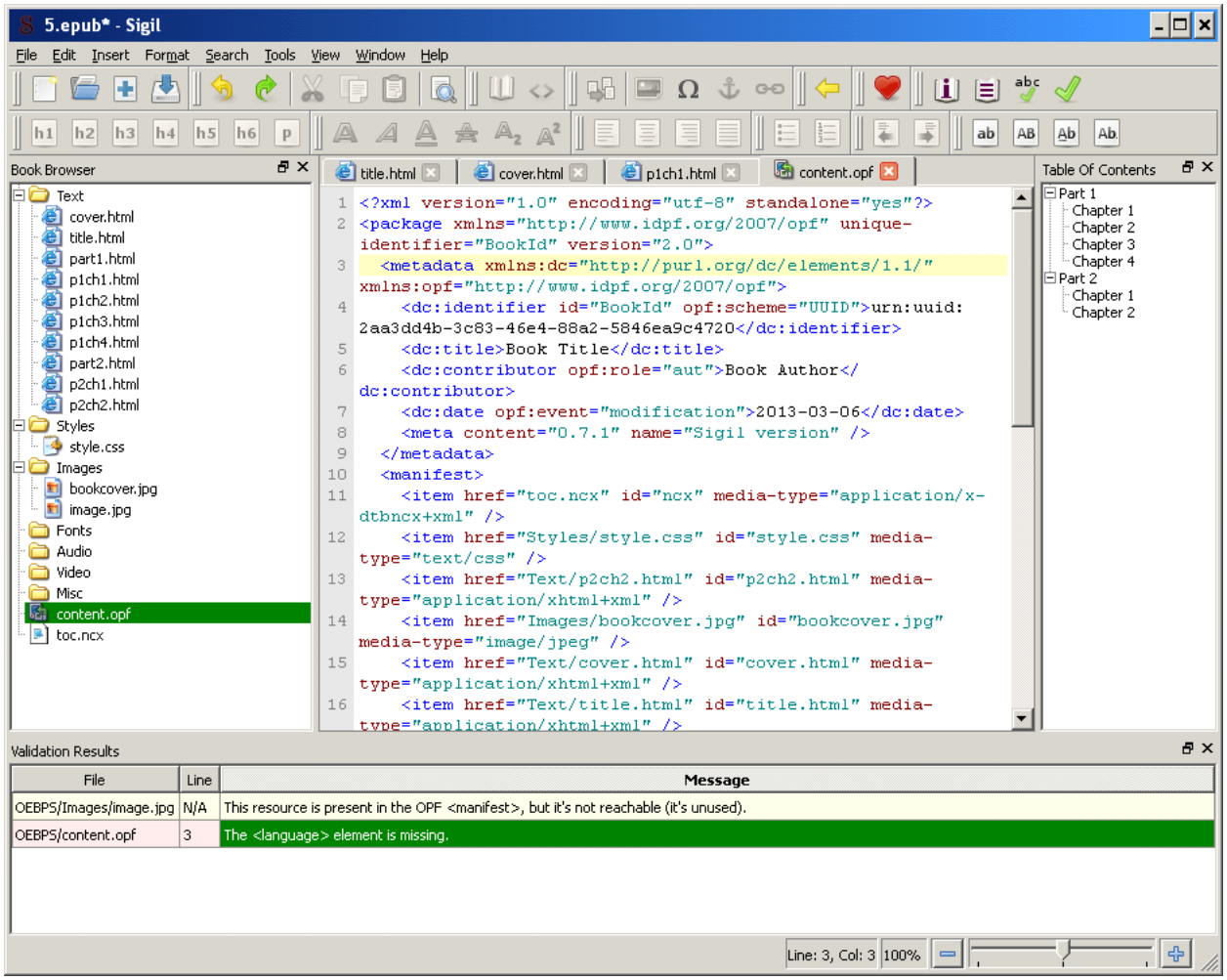
Sigil includes an internal ePub checker called “FlightCrew”. To invoke FlightCrew, just click on the “Validate ePub” icon. The results are displayed in the “Validation Results” panel at the bottom of Sigil’s window.



In the example we see two error messages. The first one tells us that there’s an unused image in the document. The second one, that we haven’t defined the language of the document.

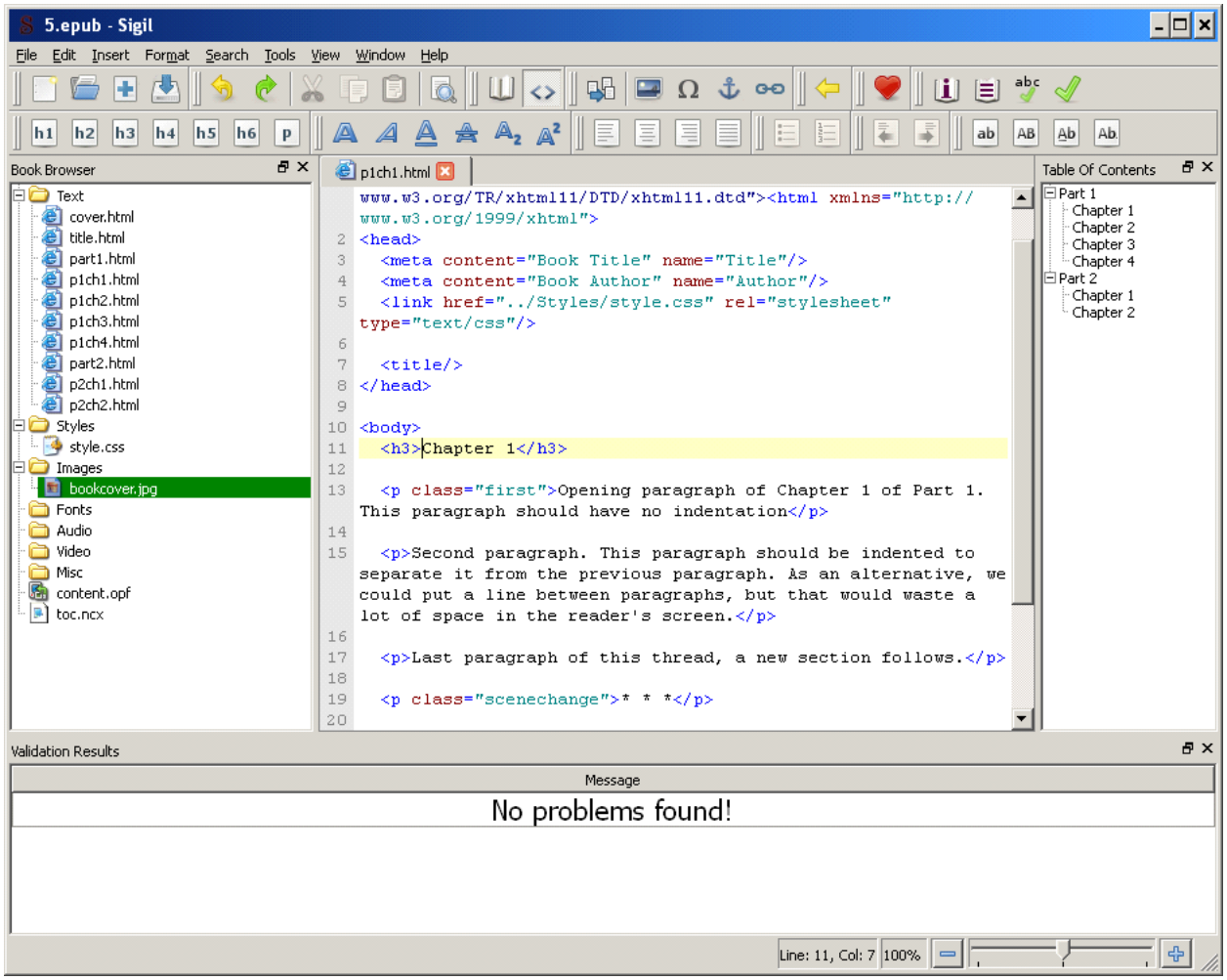
Solving these problems is quite easy: for the first one, just delete the unused image or reference it somewhere inside the text of the book. For the second one, go to the metadata editor, select the language and exit with the OK button, as mentioned before.

Double-clicking on the first error line opens a tab that displays the unused image. This is quite harmless, but clicking on the second line is not: it opens the “content.opf” file in a new tab and allows us to edit it:



For beginners, it is best to avoid direct editing of the “content.opf” and “toc.ncx” files, unless the problem cannot be solved otherwise. In this example, you can solve the error in the metadata editor and Sigil will modify the “content.opf” file accordingly.

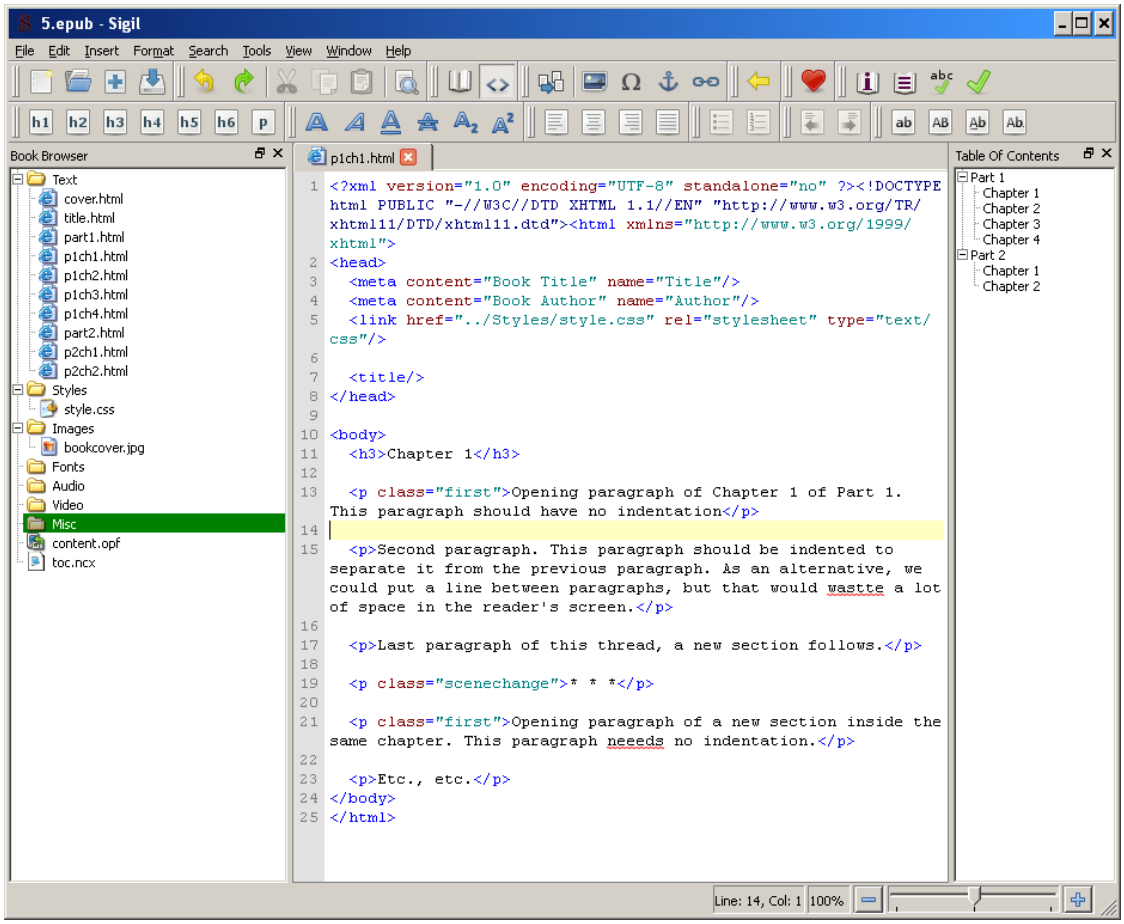
After making the necessary corrections, you have to re-invoke FlightCrew. If everything is right, you will get the following:



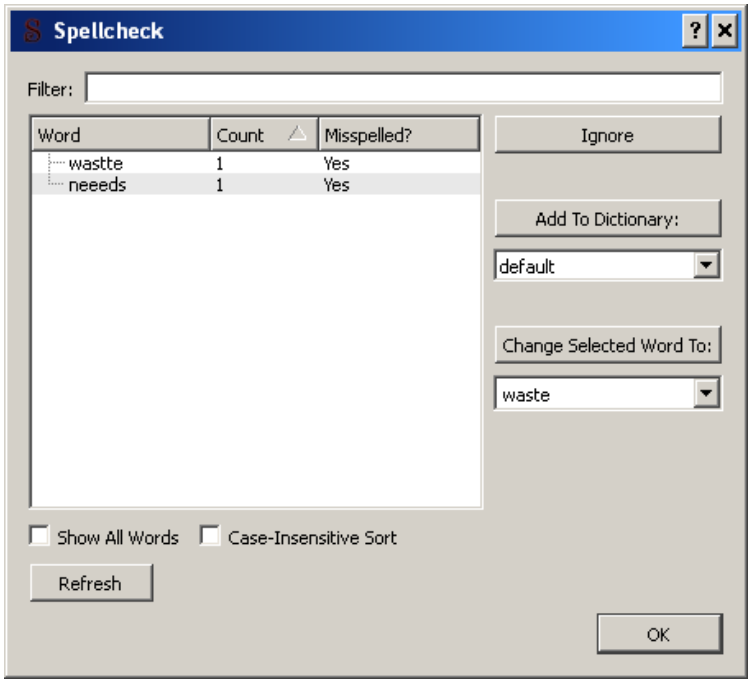
Spell checking

Sigil includes spell checking in Code View only. The dictionary used for spell checking is not taken from the language setting for the document entered in the metadata editor: you have to set in it the Preferences dialog with **Edit→Preferences→ Spellcheck Dictionaries**. Spelling errors are underlined in red in Code View (only if the "Highlight Misspelled Words" option is checked in the preferences or with **Tools→Spellcheck→Highlight Misspelled Words**).

Right-clicking on a misspelled word shows a context menu where you can choose an alternative word to replace it, ignore it or add it to a dictionary.



Clicking on the “Spellcheck” icon brings up the following dialog:



Here you can see all the misspelled words at the same time and choose what action to perform on each one of them.

For a detailed explanation of Sigil's spellchecking features, see the online user's guide at

http://web.sigil.googlecode.com/git/files/OEBPS/Text/spellcheck.html

Finding and Replacing text

Sigil provides two types of Find and Replace modes:

- Normal (case insensitive) and Case Sensitive searches of user-entered text strings.
- Regular expressions (RegEx).

The "normal" and "case sensitive" modes are suitable for very simple tasks like replacing "colour" with "color" or "parking lot" with "car park". There is no "wildcard" support in these modes, making them almost useless. The "regular expressions" mode is very useful for generic searches like bad paragraph ends (those that don't end with a period, a question, exclamation or quotation mark) and complex replacements like coding the first letter of the first word of each chapter as a drop cap.

Find and Replace can be used both in Book View and in Code View, but the usage in Book View is restricted to the current (x)html file. To search inside all files, you have to do it in Code View.

There's not much to say about Find and Replace in the "normal" and "case sensitive" modes. On the other hand, there's a lot to say about the RegEx mode, but this is an advanced topic well outside the scope of this tutorial. Learning to use RegEx mode is a must if you intend to do any serious book editing. The online user's guide only gives an introduction to this topic and provides a reference page. A good regex tutorial can be found at:

http://www.regular-expressions.info/tutorialcnt.html

Basic formatting tasks

Paragraphs

Paragraph formatting should be designed so that it is always clear to the reader where each paragraph starts and ends. One way of achieving this is by leaving a blank space after each paragraph. In this case, first-line indentation is not used. In our stylesheet:

```
p {
        text-indent: 0;
        text-align: justify;
        font-size: 1em;
        margin-top:0;
        margin-bottom: 1em;
}
```

A common error found in poorly formatted books is the use of empty paragraphs to achieve the separation instead of proper CSS coding. Another mistake is the use of both paragraph separation and first-line indentation.

The paragraph separation method wastes a lot of screen space in ebook readers. As an alternative, an indentation at the beginning of a line is often used to signal the start of a new paragraph. In this case, the first paragraph in a chapter or section is not indented. In our stylesheet:

```
p {
        text-indent:  2em;
        text-align: justify;
        font-size: 1em;
        margin-top:0;
        margin-bottom: 0;
}

.first {text-indent: 0;}
```

In the body:

```
<p class="first">First paragraph in chapter or section</p>
<p>Second paragraph</p>
<p>Third paragraph</p>
```

Indentation can be a problem in very small screens, in which case a small separation between paragraphs may work better.

Scene changes

When the paragraph separation method is used, a visual indication is needed whenever there is a change of scene in the text. For example,

```
p {
        text-indent: 0;
        text-align: justify;
        font-size: 1em;
        margin-top:0;
        margin-bottom: 1em;
}
.center {text-align: center; }

<p>Last paragraph in section 1</p>
<p class="center">* * *</p>
<p>First paragraph in section 2</p>
```

Where "* * *" signals the change of scene. Note that this indication cannot be safely replaced by increasing paragraph spacing, as the scene change could occur near the end of a page and go undetected.

If indenting is used:

```
p {
        text-indent:  2em;
        text-align: justify;
        font-size: 1em;
        margin-top:0;
        margin-bottom: 0;
}
.scenechange {text-indent: 0; text-align: center; margin-top:1em; margin-bottom: 1em;}

<p>Last paragraph in section 1</p>
<p class="scenechange">* * *</p>
<p class="first">First paragraph in section 2</p>
```

Footnotes

Real footnotes (those displayed at the bottom of the page) are not supported by the ePub standard. A workaround is to put all footnotes at the end of the book (endnotes).

Example:

In file Section0001.xhtml we have the following text:

The spanish word *gratis*[1] is sometimes used to refer to free software.

And in Section0010.xhtml

Notes
[1] **Gratis: free of charge.**

When the reader clicks, taps or navigates to [1], the notes page is shown in the screen. When he is done reading the note, he should be able to go back to where he was previously.

To link from the [1] in the body to the note, we should write something like:

In Section0001.xhtml:

```
<p>The spanish word <i>gratis</i><a href="../Text/Section0010.xhtml#note_1">[1]</a> is sometimes used to refer to free software.</p>
```

In Section0010.xhtml:

```
<h2>Notes</h2>
<p><a id="note_1"></a>[1] Gratis: free of charge.</p>
```

This is enough is the device provides a “history” button or function that lets the reader go back to the body of the text. Not all devices feature this function, so to be on the safe side, a “go back” link can be implemented:

In Section0001.xhtml:

```
<p>The spanish word <i>gratis</i><a id="goback_1"></a><a href="../Text/Section0010.xhtml#note_1">[1]</a> is sometimes used to refer to free software.</p>
```

In Section0010.xhtml:

```
<h2>Notes</h2>
<p><a id="note_1"></a><a href="../Text/Section0001.xhtml#goback_1"> [1]</a> Gratis: free of charge.</p>
```

Now the link works in both ways and there’s no need for a history function in the device.

You can manually code all your notes or you can use the “Insert ID” and “Insert Link” icons. If you prefer to use the icons, you must first mark the IDs in the notes and after that insert the links in the body.

Advanced formatting tasks

Font embedding

Each device has its own built-in fonts for displaying books. If you want to use different fonts you have to embed them and specify how they will be used in your stylesheet.

The first step is finding a font family that you like. There are tons of free fonts in the net, in our case we’ll use the Fontin family, available at http://www.exljbris.com/fontin.html. Download the true type font (TTF) and unzip the file. You’ll get 4 files:

- Fontin-Regular.ttf
- Fontin-Italic.ttf
- Fontin-Bold.ttf
- Fontin-SmallCaps.ttf

We’ll use the first three.

In Sigil, right click on the “Fonts” folder in the “Book Browser” panel and select “Add Existing Items”, browse to the place where you have stored your fonts and select them.

The next step is declaring the fonts in the stylesheet. To do that, just add the following at the top:

```
@font-face {
 font-family: "Fontin";
 font-weight: normal;
 font-style: normal;
 src: url(../Fonts/Fontin-Regular.ttf);
}

@font-face {
 font-family: "Fontin";
 font-weight: bold;
 font-style: normal;
 src: url(../Fonts/Fontin-Bold.ttf);
}

@font-face {
 font-family: "Fontin";
 font-weight: normal;
 font-style: italic;
 src: url(../Fonts/Fontin-Italic.ttf);
}
```

Now you have to decide how you will use the fonts. If you want to use the embedded font for all the text in the book, just add this line after the font declarations:

```
body {  font-family: "Fontin", serif; }
```

If you want to use the fonts just for headings, add the following line:

```
h1, h2, h3 {  font-family: "Fontin", serif; }
```

Another example:

```
p {
        font-family: "Fontin", serif;
        text-indent: 0;
        text-align: justify;
        font-size: 1em;
        margin-top:0;
        margin-bottom: 1em;
}
```

Drop Caps

Drop caps like the first letter of this paragraph are sometimes used as an embellishment at the start of each chapter. To add drop caps to your book, you must first add a new class to your stylesheet and then modify the first paragraph of each chapter as follows:

In your stylesheet:

```
span.dropcap {
font-size: 300%;
font-weight: bold;
float: left;
margin: -0.1em 0.125em -0.2em 0em;
text-indent: 0em;
line-height: 1em;
height: 1em;}
```

In the first paragraph of each chapter:

```
<p class="first"><span class="dropcap">O</span>pening paragraph of Chapter…</p>
```

Note that the “span” declaration lets you apply formatting to *a part* of an element (in this case, a paragraph), that is enclosed between and tags. It works like and to italize a fragment of text.

If the first paragraph is too short, it is not guaranteed that the second paragraph will start under the drop cap. To force the second paragraph to start under the drop cap, you have to add something like this to your stylesheet:

```
p.first+p {clear: left;}
```

This means that any paragraph just after a class “first” paragraph should start from the left.

Images

We have already inserted an image in Sigil for the cover. The same method can be used anywhere in the book, but now we’ll look at a better way to do the task.

Once we have our images ready, we have to include them in Sigil. We’ll do this by right-clicking on the “Images” folder in the “Book Browser” panel and selecting “Add Existing Items…”. The images included in this way will not show on the book yet.

In our stylesheet, we’ll add the following:

```
.ir { float: right; margin: 3px; page-break-inside: avoid;}
.il { float: left; margin: 3px; page-break-inside: avoid;}
.ic { page-break-inside: avoid; }
.caption { font-size: 0.9em; font-weight: bold; text-align: center; text-indent: 0; margin-bottom: 1em;}
```

The class selectors “ir”, “il” and “ic” are for right aligned, left aligned and centered images respectively, the “caption” selector is for adding a text line to the image.

Now we have to locate the place where we want to put our image and insert the following in Code View:

```
<div class="ir">
<h3  title="Figure 1"><img alt="" src="../Images/figure_1.jpg" /></h3>
<p class="caption">Figure 1: some text</p>
</div>
```

Note that:

- The image will be right aligned
- The image is inserted using the “img” tag. This is a self-closing tag (note the slash at the end)
- The “img” tag is inside a heading. This is not mandatory, but it allows us to include the image in the TOC if we want to. In case we include it in the TOC, the text will be that of the “title” property.
- There’s a caption below the image.

- Caption and image will always be on the same page, due to the “page-break-inside: avoid” line in the “ir” selector definition.

The tag creates a logical division inside the document. In the example above, the properties of the class selector “ir” apply to everything that is between and .

Poetry

When formatting poetry, we have to keep in mind the following:

- Stanzas should be clearly separated from each other.
- It is desirable to avoid splitting stanzas across pages. If a stanza does not fit in the remaining space in the screen, it should go to the next page. This may be a problem in small screens, though.
- Lines should be left-justified.
- If a line does not fit in the width of the screen, it should continue below, but some indication that it is not a new line is needed (for example, a small indent).
- A left margin is desirable.

To achieve this, we’ll add the following to our CSS:

```
.stanza {
        margin-top: 1em;
        margin-right: 0;
        margin-bottom: 1em;
        margin-left: 2em;
        text-align: left;
        page-break-inside: avoid;
}

.stanza p {
        padding-left: 2em;
        text-indent: -2em;
}
```

If we want to allow stanzas to split across screens, we have to delete the “page-break-inside: avoid;” declaration.
The second declaration defines the properties of paragraphs that are inside a class “stanza” division.

Now let’s code the poem “The Look” by Sara Teasdale in Code View:

```
<h1>The Look</h1>
<h2>by Sara Teasdale</h2>
<div class="stanza">
        <p> Strephon kissed me in the spring, </p>
        <p>Robin in the fall, </p>
        <p>But Colin only looked at me</p>
        <p>And never kissed at all. </p>
 </div>
<div class="stanza">
        <p>Strephon's kiss was lost in jest, </p>
        <p>Robin's lost in play, </p>
        <p>But the kiss in Colin's eyes</p>
        <p>Haunts me night and day. </p>
</div>
```

Note that each line has been defined as a paragraph. Looking in the CSS definition, we see there's a 2em left "padding" (an additional margin) for each paragraph, but this padding is cancelled by the negative text indent (-2em), so it has no effect *unless the "paragraph" doesn't fit in a line*. In this case, the second line will be indented with respect to the first one, clearly indicating that it is a continuation of the previous line and not a new line.

Letters

When formatting a letter it is a good idea to define left and right margins so that it clearly stands out from the surrounding text. Paragraphs may be separated by a line instead of using indents and the signature can be right aligned. For example, in our CSS:

```
.letter {margin-left: 3em; margin-right: 3em; }
.letter p{text-indent: 0; text-align: justify; font-size: 1em; margin-top: 1.25em;}
.letter p.sign {text-align: right;}
```

In the body of the text:

```
<div class="letter">
        <p>"Madame la Comtesse,</p>
        <p>"The Christian feelings with which your heart is filled give me the, I feel, unpardonable
        boldness to write to you. I am miserable at being separated from my son. I entreat permission to
        see him once before my departure. Forgive me for recalling myself to your memory. I apply to
        you and not to Alexey Alexandrovitch, simply because I do not wish to cause that generous man
        to suffer in remembering me. Knowing your friendship for him, I know you will understand me.
        Could you send Seryozha to me, or should I come to the house at some fixed hour, or will you let
        me know when and where I could see him away from home? I do not anticipate a refusal,
        knowing the magnanimity of him with whom it rests. You cannot conceive the craving I have to
        see him, and so cannot conceive the gratitude your help will arouse in me.</p>
        <p class="sign">Anna"</p>
</div>
```

Back to Sigil: Code View vs. Book View

It should be clear by now why almost all editing is done in Code View: there's no way in Book View to perform the formatting tasks we've discussed above.

Even the simplest actions performed in Book View can lead to some disagreeable results. For example, in Book View, hit the enter key a couple of times and change to Code View to see the code Sigil has added. You'll see a couple of hideous lines like this:

The self-closing tag
 stands for "break". This is not the correct way to introduce a blank space. If you want do that, define an appropriate class selector in your CSS and apply it to the preceding or next paragraph. Worst of all, some readers directly ignore empty paragraphs like this. Ok, you may think that adding an space solves the problem. If you do that in Book View and then change to Code View, you'll see this:

" " is a special symbol that means "non breaking space". For example, if you write

No separation

it will be treated like a single word and displayed "No separation", always in the same line (that's why it is called a *non breaking* space). Not exactly what you want to create a blank line.

Another nasty side-effect of editing in Book View is the automatic creation of inline class selectors **(only fi "Cleaning with HTML Tidy" is enabled)**. For example, in Book View, create a new paragraph and write something, then highlight it and press the Bold icon. Change to Code View. You'll find something like this:

something

And looking inside the <head> section of the file, you'll see

```
<style type="text/css">
p.sgc-1 { font-weight: bold }
</style>
```

This is a class selector declaration that is *outside your CSS*! Even if you delete what you just wrote, the declaration will remain, as there is no "garbage collection" in Sigil (you'll have to delete it manually). Another problem is that inline styles are local to the file in which they are used, so while in Section001.xhtml "sgc-1" is configured as "font-weight: bold", in Section002.xhtml it could be configured as "font-weight: italic".

In short, only use Book View to see the effect of your html/css code and for minor text corrections, like typos, spelling mistakes and the like.

Summary and acknowledgements

I learnt all I know about ebook formatting at MobileRead, reading posts, asking questions and looking inside books uploaded by the great masters Patricia, Jellby, Zelda Pinwheel, HarryT and others. Many examples used in this tutorial are modified versions of fragments extracted from Zelda's *Three men in a boat* (font embedding, images), Jellby's *Martín Fierro* (Poetry) and the thread "epub code snippets (html/css)" found at http://www.mobileread.com/forums/showthread.php?t=46448 (drop caps).

When I bought my first reader four years ago, I knew nothing about HTML and CSS, ePub was just a promise and Sigil didn't even exist. This document was inspired by my own learning process.

In this tutorial we have covered pretty much everything that is needed to make a nice looking book. I am aware that many concepts have not been explained in detail, but I just wanted to write a quick startup guide, not a book. I hope you find it useful.

Pablo