Build a digital book with EPUB

The open XML-based eBook format

Liza Daly

July 13, 2011
(First published November 25, 2008)

Need to distribute documentation, create an eBook, or just archive your favorite blog posts? EPUB is an open specification for digital books based on familiar technologies like XML, CSS, and XHTML, and EPUB files can be read on portable e-ink devices, mobile phones, and desktop computers. This tutorial explains the EPUB format in detail, demonstrates EPUB validation using Java™ technology, and moves step-by-step through automating EPUB creation using DocBook and Python.

05 Feb 2009 - *As a followup to reader comments, the author revised the content of* *Listing 3* *and refreshed the epub-raw-files.zip file (see* *Downloads*).

27 Apr 2010 - *Refreshed the epub-raw-files.zip file (see* *Downloads*).

03 Jun 2010 - *At author request,revised the content of Listings* *3* *and* *8*. *Also refreshed the epub-raw-files.zip file (see* *Downloads*).

11 Jan 2011 - *At author request,revised the content of Listing* *5*. *Changed second line of code from* `<item id="ncx" href="toc.ncx" media-type="text/xml"/>;` *to* `<item id="ncx" href="toc.ncx" media-type="application/x-dtbncx+xml"/>`.

12 Jul 2011 - *As a followup to reader comments, revised the content of* *Listing 14*. *Removed* `` ` `` *character near end of first line of code from* `<?xml version="1.0" encoding="utf-8"?`>`. *Revised code now reads:* `<?xml version="1.0" encoding="utf-8"?>`.

Before you start

This tutorial guides you through creating eBooks in the EPUB format. EPUB is an XML-based, developer-friendly format that is emerging as the de facto standard for digital books. But EPUB isn't just for books: With it, you can:

- Bundle documentation for offline reading or easy distribution
- Package blog posts or other Web-native content

- Build, search, and remix using common open source tools

About this tutorial

Frequently used acronyms

- API: application programming interface
- CSS: Cascading stylesheets
- DOM: Document Object Model
- DTD: Document type definition
- GUI: Graphical user interface
- HTML: Hypertext Markup Language
- SAX: Simple API For XML
- W3C: World Wide Web Consortium
- XHTML: Extensible HTML
- XML: Extensible Markup Language

You start this tutorial by generating an EPUB book manually to help you learn all the components and required files. Next, the tutorial shows how to bundle the finished digital book and validate it against the specification as well as how to test it in various reading systems.

Then, it covers generating EPUB from DocBook XML—one of the most widely used standards for technical documentation—and how to use Python to completely automate EPUB creation with DocBook from end to end.

Objectives

In this tutorial, you:

- Learn what EPUB is, who's behind it, and who's adopting it
- Explore the structure of an EPUB bundle, including its required files and their schemas
- Create a valid EPUB file from scratch using simple content
- Use open source tools to produce EPUB files from DocBook, a widely used schema for technical documentation and books
- Automate EPUB conversion using Python and DocBook

Prerequisites

No particular operating system is assumed for this tutorial, although you should be familiar with the mechanics of creating files and directories. Use of an XML editor or integrated development environment (IDE) is strongly recommended.

For the later parts of the tutorial on automating EPUB creation, this tutorial assumes that you know one or more basic XML processing skills—XSLT, DOM, or SAX-based parsing—and how to construct an XML document using an XML-native API.

No familiarity with the EPUB file format is necessary to complete this tutorial.

System requirements

To complete the examples in this tutorial, you need a Java interpreter (version 1.5 or later) and a Python interpreter (version 2.4 or later) as well as the required XML libraries for each. However,

experienced XML developers should be able to adapt the examples to any programming language with XML libraries.

About the EPUB format

Learn the background of EPUB, what EPUB is most commonly used for, and how EPUB differs from the Portable Document Format (PDF).

What is EPUB?

EPUB is the XML format for reflowable digital books and publications standardized by the International Digital Publishing Forum (IDPF), a trade and standards association for the digital publishing industry. IDPF officially adopted EPUB in October 2007 and by 2008 had seen rapid adoption by major publishers. You can read the EPUB format using a variety of open source and commercial software on all major operating systems, e-ink devices such as the Sony PRS, and small devices such as the Apple iPhone.

Who is producing EPUB? Is it only for books?

Although traditional print publishers were the first to adopt EPUB, nothing in the format limits its use to eBooks. With freely available software tools, you can bundle Web pages as EPUB, convert plain text files, or transform existing DocBook XML documentation into well-formed and valid EPUB. (I cover the latter in From DocBook to EPUB.)

How is EPUB different from PDF?

PDF is still the most widely used electronic document format in the world. From a book publisher's point of view, PDF has several advantages:

- PDF files allow pixel-perfect control over layout, including complex print-friendly layouts such as multiple columns and alternate recto/verso styles.
- PDFs can be generated by a wide variety of GUI-based document tools, such as Microsoft® Office Word or Adobe® InDesign®.
- PDF readers are ubiquitous and installed on most modern computers.
- Specific fonts can be embedded in PDF to control the final output exactly.

> ## Three standards in one
>
> EPUB consists of three separate IDPF specifications, although in practice, it's safe to refer to them collectively as *EPeUB*:
>
> - **Open eBook Publication Structure Container Format (OCF):** Specifies the directory tree structure and file format (ZIP) of an EPUB archive.
> - **Open Publication Structure (OPS):** Defines the common vocabularies for the eBook, especially the formats allowed to be used for book content (for example, XHTML and CSS).
> - **Open Packaging Format (OPF):** Describes the required and optional metadata, reading order, and table of contents in an EPUB.

> Additionally, EPUB reuses several other standards, such as XHTML version 1.0 and Digital Accessible Information SYstem (DAISY), for specific types of content within the EPUB archive.

From a software developer's point of view, PDF falls far short of the ideal:

- It's not a trivial standard to learn; therefore, it's not a simple matter to throw together your own PDF-generating code.
- Although PDF is now an International Organization for Standardization (ISO) standard (ISO 32000-1:2008), traditionally it has been controlled by a single corporation: Adobe Systems.
- Although PDF libraries are available for most programming languages, many are commercial or are embedded in GUI applications and not easily controlled by external processes. Not all free libraries continue to be actively maintained.
- PDF-native text can be extracted and searched programmatically, but few PDFs are tagged such that conversion to a Web-friendly format is simple or reliable.
- PDF documents aren't easily *reflowable*, meaning that they don't adapt well to small screens or to radical changes to their layouts.

Why EPUB is friendly to developers

EPUB addresses all the flaws in PDF as they relate to developer friendliness. An EPUB is a simple ZIP-format file (with an *.epub* extension) that contains files ordered in a proscribed manner. There are a few tricky requirements about how the ZIP archive is prepared, which will be discussed in detail later in Bundling your EPUB file as a ZIP archive. Otherwise, EPUB is simple:

- Nearly everything in EPUB is XML. EPUB files can be built using standard XML toolkits without any special or proprietary software.
- EPUB content (the actual text of an eBook) is almost always XHTML version 1.1. (An alternative format is DTBook, a standard for encoding books for the visually impaired. See Related topics for more information on DTBook, which is not covered in this tutorial).
- Most of the EPUB XML schemas are taken from existing, published specifications that are freely available.

The two key points are that **EPUB metadata is XML** and **EPUB content is XHTML**. If your documentation-building system produces output for the Web and/or is based on XML, then it is very close to being able to produce EPUB, as well.

Building your first EPUB

A minimally conforming EPUB bundle has several required files. The specification can be quite strict about the format, contents, and location of those files within the EPUB archive. This section explains what you *must* know when you work with the EPUB standard.

Anatomy of an EPUB bundle

The basic structure of a minimal EPUB file follows the pattern in Listing 1. When ready for distribution, this directory structure is bundled together into a ZIP-format file, with a few special requirements discussed in Bundling your EPUB file as a ZIP archive.

Listing 1. Directory and file layout for a simple EPUB archive

```
mimetype
META-INF/
   container.xml
OEBPS/
  content.opf
  title.html
  content.html
  stylesheet.css
  toc.ncx
  images/
     cover.png
```

Note: A sample book following this pattern is available from Downloadable resources, but I recommend that you create your own as you follow the tutorial.

To start building your EPUB book, create a directory for the EPUB project. Open a text editor or an IDE such as Eclipse. I recommend using an editor that has an XML mode—in particular, one that can validate against the Relax NG schemas listed in Related topics.

The mimetype file

This one's pretty easy: The mimetype file is required and must be named *mimetype.* The contents of the file are always:

```
application/epub+zip
```

Note that the mimetype file cannot contain any newlines or carriage returns.

Additionally, the mimetype file must be the first file in the ZIP archive and must not itself be compressed. You'll see how to include it using common ZIP arguments in Bundling your EPUB file as a ZIP archive. For now, just create this file and save it, making sure that it's at the root level of your EPUB project.

META-INF/container.xml

At the root level of the EPUB, there must be a META-INF directory, and it must contain a file named *container.xml.* EPUB reading systems will look for this file first, as it points to the location of the metadata for the digital book.

Create a directory called *META-INF*. Inside it, open a new file called *container.xml* for writing. The container file is very small, but its structural requirements are strict. Paste the code in Listing 2 into META-INF/container.xml.

Listing 2. Sample container.xml file

```
<?xml version="1.0"?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <rootfiles>
    <rootfile full-path="OEBPS/content.opf"
    media-type="application/oebps-package+xml" />
  </rootfiles>
</container>
```

The value of `full-path` (in bold) is the only part of this file that will ever vary. The directory path must be relative to the root of the EPUB file itself, not relative to the META-INF directory.

> **More about META-INF**
>
> The META-INF directory can contain a few optional files, as well. These files allow EPUB to support digital signatures, encryption, and digital rights management (DRM). These topics are not covered in this tutorial. See the OCF specification for more information.

The mimetype and container files are the only two whose location in the EPUB archive are strictly controlled. As recommended (although not required), store the remaining files in the EPUB in a sub-directory. (By convention, this is usually called *OEBPS,* for *Open eBook Publication Structure,* but can be whatever you like.)

Next, create the directory named *OEBPS* in your EPUB project. The following section of this tutorial covers the files that go into OEBPS—the real meat of the digital book: its metadata and its pages.

Open Packaging Format metadata file

Although this file can be named anything, the OPF file is conventionally called *content.opf.* It specifies the location of *all* the content of the book, from its text to other media such as images. It also points to another metadata file, the Navigation Center eXtended (NCX) table of contents.

The OPF file is the most complex metadata in the EPUB specification. Create OEBPS/content.opf, and paste the contents of Listing 3 into it.

Listing 3. OPF content file with sample metadata

```
<?xml version='1.0' encoding='utf-8'?>
<package xmlns="http://www.idpf.org/2007/opf"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         unique-identifier="bookid" version="2.0">
  <metadata>
    <dc:title>Hello World: My First EPUB</dc:title>
    <dc:creator>My Name</dc:creator>
    <dc:identifier
id="bookid">urn:uuid:0cc33cbd-94e2-49c1-909a-72ae16bc2658</dc:identifier>
    <dc:language>en-US</dc:language>
    <meta name="cover" content="cover-image" />
  </metadata>
  <manifest>
    <item id="ncx" href="toc.ncx" media-type="application/x-dtbncx+xml"/>
    <item id="cover" href="title.html" media-type="application/xhtml+xml"/>
    <item id="content" href="content.html"
media-type="application/xhtml+xml"/>
    <item id="cover-image" href="images/cover.png" media-type="image/png"/>
    <item id="css" href="stylesheet.css" media-type="text/css"/>
  </manifest>
  <spine toc="ncx">
    <itemref idref="cover" linear="no"/>
    <itemref idref="content"/>
  </spine>
  <guide>
    <reference href="title.html" type="cover" title="Cover"/>
```

```
   </guide>
</package>
```

OPF schemas and namespaces

The OPF document itself must use the namespace *http://www.idpf.org/2007/opf,* and the metadata will be in the Dublin Core Metadata Initiative (DCMI) namespace, *http://purl.org/dc/elements/1.1/.*

This would be a good time to add the OPF and DCMI schema to your XML editor. All the schemas used in EPUB are available from Downloadable resources.

Metadata

Dublin Core defines a set of common metadata terms that you can use to describe a wide variety of digital materials; it's not part of the EPUB specification itself. Any of these terms are allowed in the OPF metadata section. When you build an EPUB for distribution, include as much detail as you can here, although the extract provided in Listing 4 is sufficient to start.

Listing 4. Extract of OPF metadata

```
...
<metadata>
  <dc:title>Hello World: My First EPUB</dc:title>
  <dc:creator>My Name</dc:creator>
  <dc:identifier id="bookid">urn:uuid:12345</dc:identifier>
  <meta name="cover" content="cover-image" />
</metadata>
...
```

The two required terms are *title* and *identifier.* According to the EPUB specification, the identifier *must* be a unique value, although it's up to the digital book creator to define that unique value. For book publishers, this field will typically contain an ISBN or Library of Congress number. For other EPUB creators, consider using a URL or a large, randomly generated unique user ID (UUID). Note that the value of the attribute `unique-identifier` must match the ID attribute of the `dc:identifier` element.

Other metadata to consider adding, if it's relevant to your content, include:

- Language (as `dc:language`).
- Publication date (as `dc:date`).
- Publisher (as `dc:publisher`). (This can be your company or individual name.)
- Copyright information (as `dc:rights`). (If releasing the work under a Creative Commons license, put the URL for the license here.)

See Related topics for more information on DCMI.

Including a `meta` element with the `name` attribute containing `cover` is not part of the EPUB specification directly, but is a recommended way to make cover pages and images more portable. Some EPUB renderers prefer to use an image file as the cover, while others will use an XHTML file containing an inlined cover image. This example shows both forms. The value of the `meta`

element's `content` attribute should be the ID of the book's cover image in the manifest, which is the next part of the OPF file.

Manifest

The OPF manifest lists all the resources found in the EPUB that are part of the content (and excluding metadata). This usually means a list of XHTML files that make up the text of the eBook plus some number of related media such as images. EPUB encourages the use of CSS for styling book content, so CSS files are also included in the manifest. **Every file that goes into your digital book must be listed in the manifest.**

[Listing 5](#) shows the extracted manifest section.

Listing 5. Extract of OPF manifest

```
...
<manifest>
  <item id="ncx" href="toc.ncx" media-type="application/x-dtbncx+xml"/>
  <item id="cover" href="title.html" media-type="application/xhtml+xml"/>
  <item id="content" href="content.html" media-type="application/xhtml+xml"/>
  <item id="cover-image" href="images/cover.png" media-type="image/png"/>
  <item id="css" href="stylesheet.css" media-type="text/css"/>
</manifest>
...
```

Advanced OPF manifests

A more advanced sample of a manifest file will include multiple XHTML files as well as images and a CSS. Get a complete EPUB with examples of common types from [Downloadable resources](#).

You must include the first item, `toc.ncx` (discussed in the [next section](#)). Note that all items have an appropriate `media-type` value and that the media type for the XHTML content is `application/xhtml+xml`. This exact value is required and *cannot* be `text/html` or some other type.

EPUB supports four image file formats as *core* types: Joint Photographic Experts Group (JPEG), Portable Network Graphics (PNG), Graphics Interchange Format (GIF), and Scalable Vector Graphics (SVG). You can include non-supported file types if you provide a fall-back to a core type. See the OPF specification for more information on fall-back items.

The values of the `href` attribute should be a Uniform Resource Identifier (URI) that is relative *to the OPF file.* (This is easy to confuse with the reference to the OPF file in the container.xml file, where it must be relative to the EPUB as a whole.) In this case, the OPF file is in the same OEBPS directory as your content, so no path information is required here.

Spine

Although the manifest tells the EPUB reader which files are part of the archive, the spine indicates the order in which they appear, or—in EPUB terms—the *linear reading order* of the digital book. One way to think of the OPF spine is that it defines the order of the "pages" of the book. The spine is read in document order, from top to bottom. [Listing 6](#) shows an extract from the OPF file.

Listing 6. Extract of OPF spine

```
...
<spine toc="ncx">
  <itemref idref="cover" linear="no"/>
  <itemref idref="content"/>
</spine>
...
```

Each `itemref` element has a required attribute `idref`, which must match one of the IDs in the manifest. The `toc` attribute is also required. It references an ID in the manifest that must indicate the file name of the NCX table of contents.

The `linear` attribute in the spine indicates whether the item is considered part of the linear reading order versus being extraneous front- or end-matter. I recommend that you define any cover page as `linear=no`. Conforming EPUB reading systems will open the book to the first item in the spine that's *not* set as `linear=no`.

Guide

The last part of the OPF content file is the guide. This section is optional but recommended. Listing 7 shows an extract from a guide file.

Listing 7. Extract of an OPF guide

```
...
<guide>
  <reference href="cover.html" type="cover" title="Cover"/>
</guide>
...
```

The guide is a way of providing semantic information to an EPUB reading system. While the manifest defines the physical resources in the EPUB and the spine provides information about their order, the guide explains what the sections mean. Here's a partial list of the values that are allowed in the OPF guide:

- **`cover:`** The book cover
- **`title-page:`** A page with author and publisher information
- **`toc:`** The table of contents

For a complete list, see the OPF 2.0 specification, available from Related topics.

NCX table of contents

Overlap between NCX and OPF metadata

Because the NCX is borrowed from another standard, there is some overlap between the information encoded in the NCX and that in the OPF. This is rarely a problem when you generate EPUBs programmatically, where the same code can output to two different files. Take care to put the same information in both places, as different EPUB readers might use the values from one or the other.

Although the OCF file is defined as part of EPUB itself, the last major metadata file is borrowed from a different digital book standard. DAISY is a consortium that develops data formats for

readers who are unable to use traditional books, often because of visual impairments or the inability to manipulate printed works. EPUB has borrowed DAISY's NCX DTD. The NCX defines the table of contents of the digital book. In complex books, it is typically hierarchical, containing nested parts, chapters, and sections.

Using your XML editor, create OEBPS/toc.ncx, and include the code in Listing 8.

Listing 8. Simple NCX file

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE ncx PUBLIC "-//NISO//DTD ncx 2005-1//EN"
                "http://www.daisy.org/z3986/2005/ncx-2005-1.dtd">
<ncx xmlns="http://www.daisy.org/z3986/2005/ncx/" version="2005-1">
  <head>
    <meta name="dtb:uid"
content="urn:uuid:0cc33cbd-94e2-49c1-909a-72ae16bc2658"/>
    <meta name="dtb:depth" content="1"/>
    <meta name="dtb:totalPageCount" content="0"/>
    <meta name="dtb:maxPageNumber" content="0"/>
  </head>
  <docTitle>
    <text>Hello World: My First EPUB</text>
  </docTitle>
  <navMap>
    <navPoint id="navpoint-1" playOrder="1">
      <navLabel>
        <text>Book cover</text>
      </navLabel>
      <content src="title.html"/>
    </navPoint>
    <navPoint id="navpoint-2" playOrder="2">
      <navLabel>
        <text>Contents</text>
      </navLabel>
      <content src="content.html"/>
    </navPoint>
  </navMap>
</ncx>
```

NCX metadata

The DTD requires four `meta` elements inside the NCX `<head>` tag:

- **uid:** Is the unique ID for the digital book. This element should match the `dc:identifier` in the OPF file.
- **depth:** Reflects the level of the hierarchy in the table of contents. This example has only one level, so this value is **1**.
- **totalPageCount** and **maxPageNumber:** Apply only to paper books and can be left at **0**.

The contents of `docTitle/text` is the title of the work, and matches the value of `dc:title` in the OPF.

NCX navMap

What's the difference between the NCX and the OPF spine?

It's okay to be confused, as both files describe the order and contents of the document. The easiest way to explain the difference is through analogy with a printed book: The OPF spine

> describes how the sections of the book are physically bound together, such that turning a page at the end of one chapter reveals the first page of the second chapter. The NCX describes the table of contents at the beginning of the book. The table of contents always includes all the major sections of the book, but it might also list sub-sections that don't occur on their own pages.
>
> A good rule of thumb is that the NCX often contains more `navPoint` elements than there are `itemref` elements in the OPF spine. In practice, all the items in the spine appear in the NCX, but the NCX can be more granular than the spine.

The `navMap` is the most important part of the NCX file, as it defines the table of contents for the actual book. The `navMap` contains one or more `navPoint` elements. Each `navPoint` must contain the following elements:

- A `playOrder` attribute, which reflects the reading order of the document. This follows the same order as the list of `itemref` elements in the OPF spine.
- A `navLabel/text` element, which describes the title for this section of the book. This is typically a chapter title or number, such as "Chapter One," or—as in this example—"Cover page."
- A `content` element whose `src` attribute points to the physical resource containing the content. This will be a file declared in the OPF manifest. (It is also acceptable to use fragment identifiers here to point to anchors within XHTML content—for example, `content.html#footnote1`.)
- Optionally, one or more child `navPoint` elements. Nested points are how hierarchical documents are expressed in the NCX.

The structure of the sample book is simple: It has only two pages, and they are not nested. That means that you'll have two `navPoint` elements with ascending `playOrder` values, starting at **1**. In the NCX, you have the opportunity to name these sections, allowing readers to jump into different parts of the eBook.

Adding the final content

Now you know all the metadata required in EPUB, so it's time to put in the actual book content. You can use the sample content provided in Downloadable resources or create your own, as long as the file names match the metadata.

Next, create these files and folder:

- **title.html:** This file will be the title page for the book. Create this file and include an `img` element that references a cover image, with the value of the `src` attribute as `images/cover.png`.
- **images:** Create this folder inside OEBPS, then copy the sample image (or create your own), naming it *cover.png.*
- **content.html:** This will be the actual text of the book.
- **stylesheet.css:** Place this file in the same OEBPS directory as the XHTML files. This file can contain any CSS declarations you like, such as setting the font-face or text color. See Listing 10 for an example of such a CSS file.

XHTML and CSS in an EPUB book

Listing 9 contains an example of a valid EPUB content page. Use this sample for your title page (title.html) and a similar one for the main content page (content.html) of your book.

Listing 9. Sample title page (title.html)

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Hello World: My First EPUB</title>
    <link type="text/css" rel="stylesheet" href="stylesheet.css" />
  </head>
  <body>
    <h1>Hello World: My First EPUB</h1>
    <div><img src="images/cover.png" alt="Title page"/></div>
  </body>
</html>
```

XHTML content in EPUB follows a few rules that might be unfamiliar to you from general Web development:

- **The content must validate as XHTML 1.1:** The only significant difference between XHTML 1.0 Strict and XHTML 1.1 is that the `name` attribute has been removed. (Use IDs to refer to anchors within content.)
- **`img` elements can only reference images that are local to the eBook:** The elements cannot reference images on the Web.
- **`script` blocks should be avoided:** There is no requirement for EPUB readers to support JavaScript code.

There are some minor differences in the way EPUB supports CSS, but none that affect common uses of styles (consult the OPS specification for details). Listing 10 demonstrates a simple CSS file that you can apply to the content to set basic font guidelines and to color headings in red.

Listing 10. Sample styles for the eBook (stylesheet.css)

```
body {
  font-family: sans-serif;
}
h1,h2,h3,h4 {
  font-family: serif;
  color: red;
}
```

One point of interest is that EPUB specifically supports the CSS 2 `@font-face` rule, which allows for embedded fonts. If you create technical documentation, this is probably not relevant, but developers who build EPUBs in multiple languages or for specialized domains will appreciate the ability to specify exact font data.

You now have everything you need to create your first EPUB book. In the next section, you'll bundle the book according to the OCF specifications and find out how to validate it.

Package and check your EPUB

By this point, you should have an EPUB bundle ready to package. This bundle will either be a new book that you created yourself or one that uses the raw files available from Downloadable resources.

Bundling your EPUB file as a ZIP archive

The OEBPS Container Format portion of the EPUB specification has several things to say about EPUB and ZIP, but the most important are:

- The first file in the archive **must** be the mimetype file (see Mimetype in this tutorial). The mimetype file must not be compressed. This allows non-ZIP utilities to uncover the mimetype by reading the raw bytes starting from position 30 in the EPUB bundle.
- The ZIP archive cannot be encrypted. EPUB supports encryption but not at the level of the ZIP file.

Using ZIP version 2.3 under a UNIX®-like operating system, create the EPUB ZIP file in two commands, as in Listing 11. (These commands assume that your current working directory is your EPUB project.)

Listing 11. Bundling the EPUB into a valid epub+zip file

```
$ zip -0Xq  my-book.epub mimetype
$ zip -Xr9Dq my-book.epub *
```

In the first command, you create the new ZIP archive and add the mimetype file with no compression. In the second, you add the remaining items. The flags `-X` and `-D` minimize extraneous information in the .zip file; `-r` will recursively include the contents of META-INF and OEBPS directories.

EPUB validation

Although the EPUB standard isn't especially difficult, its XML files must be validated against specific schemas. If you use a schema-aware XML editor to generate the metadata and XHTML, you're over halfway there. Make a final check with the EpubCheck package (see Related topics).

Adobe maintains the EpubCheck package, and it's available as open source under the Berkeley Software Distribution (BSD) license. It is a Java program that can be run as a stand-alone tool or as a Web application, or you can integrate it into an application running under the Java Runtime Environment (JRE) version 1.5 or later.

Running it from the command line is simple. Listing 12 provides an example.

Listing 12. Running the EpubCheck utility

```
$ java -jar /path/to/epubcheck.jar my-book.epub
```

If you failed to create some of the auxiliary files or introduced an error into the metadata files, you might get an error message like that in Listing 13.

Listing 13. Sample errors from EpubCheck

```
my-book.epub: image file OEBPS/images/cover.png is missing
my-book.epub: resource OEBPS/stylesheet.css is missing
my-book.epub/OEBPS/title.html(7): 'OEBPS/images/cover.png':
    referenced resource missing in the package

Check finished with warnings or errors!
```

You might need to set your CLASSPATH here to point to the location of the EpubCheck installation, as it does have some external libraries to import. You probably need to set the CLASSPATH if you get a message like:

```
org.xml.sax.SAXParseException: no implementation available for schema language
   with namespace URI "http://www.ascc.net/xml/schematron"
```

If the validation was successful, you'll see "No errors or warnings detected." In that case, congratulations on producing your first EPUB!

EPUB viewing

Testing isn't just about validation: It's also about making sure the book looks right. Do the stylesheets work properly? Are the sections actually in the correct logical order? Does the book include all the expected content?

Several EPUB readers are available that you can use for testing. Figure 1 shows a screen capture from Adobe Digital Editions (ADE), the most commonly used EPUB reader.

Figure 1. The EPUB in ADE
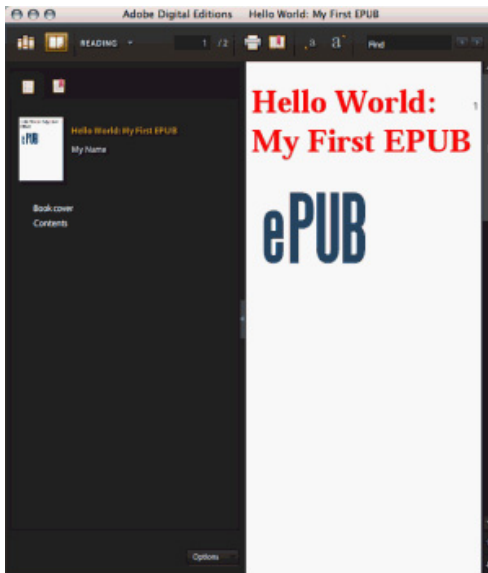
Your font colors and images are appearing, which is good. ADE is not correctly rendering the title in a sans-serif font, though, which might be a problem with the CSS. It's useful here to check in another reader. Figure 2 shows the same book rendered in my open source, Web-based EPUB reader, Bookworm.

Figure 2. The EPUB in Bookworm
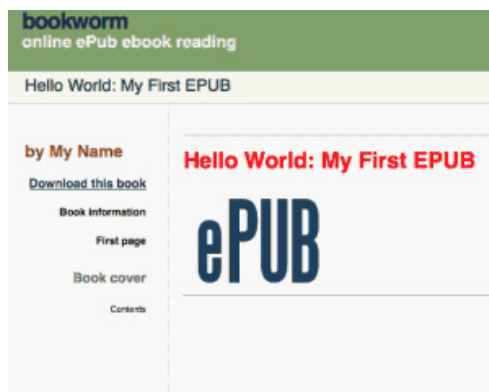
In this case, it's just that ADE doesn't support that particular declaration. Knowledge of the quirks in individual reading software will be essential if exact formatting is important in your digital book.

Now that you've done the laborious process of creating a simple EPUB from scratch, see what it takes to convert DocBook, a common XML documentation schema, into EPUB.

From DocBook to EPUB

DocBook is a popular choice for developers who need to maintain long-form technical documentation. Unlike the files produced by traditional word-processing programs, you can manage DocBook output with text-based version-control systems. Because DocBook is XML, you can easily transform it into multiple output formats. Since the summer of 2008, you can find native support of EPUB as an output format from the official DocBook XSL project.

Running the basic DocBook-to-EPUB pipeline with XSLT

Start with a simple DocBook document, in Listing 14. This document is defined as type `book` and includes a preface, two chapters, and an inline image displayed on the title page. This image will be found in the same directory as the DocBook source file. Create this file and the title page image yourself, or download samples from Downloadable resources.

Listing 14. A simple DocBook book

```
<?xml version="1.0" encoding="utf-8"?>
<book>
  <bookinfo>
    <title>My EPUB book</title>
    <author><firstname>Liza</firstname>
            <surname>Daly</surname></author>
    <volumenum>1234</volumenum>
  </bookinfo>
  <preface id="preface">
    <title>Title page</title>
    <figure id="cover-image">
      <title>Our EPUB cover image icon</title>
      <graphic fileref="cover.png"/>
    </figure>
  </preface>
  <chapter id="chapter1">
    <title>This is a pretty simple DocBook example</title>
    <para>
      Not much to see here.
    </para>
```

```
    </chapter>
    <chapter id="end-notes">
      <title>End notes</title>
      <para>
        This space intentionally left blank.
      </para>
    </chapter>
</book>
```

Next, see Related topics to download the latest version of the DocBook XSL stylesheets, and make sure that you have an XSLT processor such as xsltproc or Saxon installed. This example uses xsltproc, which is available on most UNIX-like systems. To convert the DocBook file, just run that file against the EPUB module included in DocBook XSL, as in Listing 15.

Listing 15. Converting DocBook into EPUB

```
$ xsltproc /path/to/docbook-xsl-1.74.0/epub/docbook.xsl docbook.xml
Writing OEBPS/bk01-toc.html for book
Writing OEBPS/pr01.html for preface(preface)
Writing OEBPS/ch01.html for chapter(chapter1)
Writing OEBPS/ch02.html for chapter(end-notes)
Writing OEBPS/index.html for book
Writing OEBPS/toc.ncx
Writing OEBPS/content.opf
Writing META-INF/container.xml
```

Customizing DocBook XSL

The DocBook-to-EPUB conversion pipeline is still relatively new, and you might need to customize the XSLT to get the desired output.

Next, add the mimetype file and build the epub+zip archive yourself. Listing 16 shows those three quick commands and the result of a pass through the EpubCheck validator.

Listing 16. Creating the EPUB archive from DocBook

```
$ echo "application/epub+zip" > mimetype
$ zip -0Xq  my-book.epub mimetype
$ zip -Xr9D my-book.epub *
$ java -jar epubcheck.jar my-book.epub
No errors or warnings detected
```

Pretty easy! Figure 3 shows your creation in ADE.

Figure 3. Converted DocBook EPUB in ADE
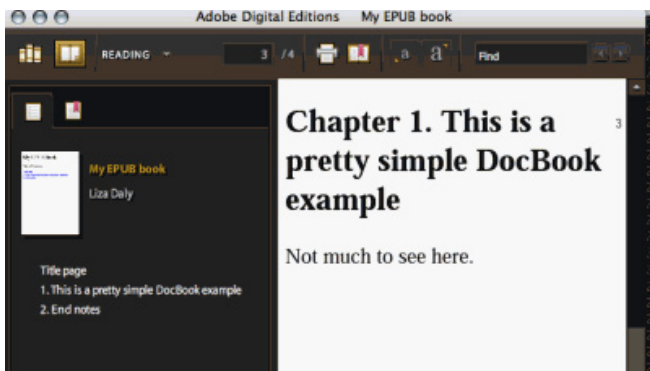
Automatic DocBook-to-EPUB conversion with Python and lxml

The DocBook XSL goes a long way toward making EPUB generation painless, but you must perform a few steps outside XSLT. This last section demonstrates a sample Python program that completes the creation of a valid EPUB bundle. I show individual methods in the tutorial; you can get the complete docbook2epub.py program from Downloadable resources.

Several Python XSLT libraries are available, but my preference is lxml. It provides not just XSLT 1.0 functionality but also high-performance parsing, full XPath 1.0 support, and special extensions for handling HTML. If you prefer a different library or use a different programming language than Python, these examples should be easy to adapt.

Calling the DocBook XSL with lxml

The most efficient method to call XSLT using lxml is to parse the XSLT in advance, then create a transformer for repeated use. This is useful, as my DocBook-to-EPUB script accepts multiple DocBook files to convert. Listing 17 demonstrates this approach.

Listing 17. Running the DocBook XSL using lxml

```
import os.path
from lxml import etree

def convert_docbook(docbook_file):
    docbook_xsl = os.path.abspath('docbook-xsl/epub/docbook.xsl')
    # Give the XSLT processor the ability to create new directories
    xslt_ac = etree.XSLTAccessControl(read_file=True,
                                      write_file=True,
                                      create_dir=True,
                                      read_network=True,
                                      write_network=False)
    transform = etree.XSLT(etree.parse(docbook_xsl), access_control=xslt_ac)
    transform(etree.parse(docbook_file))
```

The EPUB module in DocBook XSL creates the output files itself, so nothing is returned from the evaluation of the transform here. Instead, DocBook creates two folders (META-INF and OEBPS) in the current working directory that contain the results of the conversion.

Copying the images and other resources into the archive

DocBook XSL does nothing about any images that you might supply for use with your document; it only creates the metadata files and the rendered XHTML. Because the EPUB specification requires that all resources be listed in the content.opf manifest, you can inspect the manifest to find any images that were referenced in the original DocBook file. Listing 18 shows this technique, which assumes that the `path` variable contains the path to your EPUB-in-progress, as created by the DocBook XSLT.

Listing 18. Parse the OPF content file to find any missing resources

```
import os.path, shutil
from lxml import etree

def find_resources(path='/path/to/our/epub/directory'):
    opf = etree.parse(os.path.join(path, 'OEBPS', 'content.opf'))

    # All the opf:item elements are resources
    for item in opf.xpath('//opf:item',
                          namespaces= { 'opf': 'http://www.idpf.org/2007/opf' }):

        # If the resource was not already created by DocBook XSL itself,
        # copy it into the OEBPS folder
        href = item.attrib['href']
        referenced_file = os.path.join(path, 'OEBPS', href):
        if not os.path.exists(referenced_file):
            shutil.copy(href, os.path.join(path, 'OEBPS'))
```

Creating the mimetype file automatically

DocBook XSL won't create your mimetype file, either, but a quick bit of code from Listing 19 can take care of that.

Listing 19. Create the mimetype file

```
def create_mimetype(path='/path/to/our/epub/directory'):
    f = '%s/%s' % (path, 'mimetype')
    f = open(f, 'w')
    # Be careful not to add a newline here
    f.write('application/epub+zip')
    f.close()
```

Creating the EPUB bundle with Python

All that's left now is to bundle the files into a valid EPUB ZIP archive. This takes two steps: adding the mimetype file as the first in the archive with no compression, then adding the remaining directories. Listing 20 shows the code for this process.

Listing 20. Using the Python zipfile module to create an EPUB bundle

```
import zipfile, os

def create_archive(path='/path/to/our/epub/directory'):
    '''Create the ZIP archive.  The mimetype must be the first file in the archive
    and it must not be compressed.'''

    epub_name = '%s.epub' % os.path.basename(path)

    # The EPUB must contain the META-INF and mimetype files at the root, so
    # we'll create the archive in the working directory first and move it later
    os.chdir(path)

    # Open a new zipfile for writing
    epub = zipfile.ZipFile(epub_name, 'w')

    # Add the mimetype file first and set it to be uncompressed
    epub.write(MIMETYPE, compress_type=zipfile.ZIP_STORED)
```

```
    # For the remaining paths in the EPUB, add all of their files
    # using normal ZIP compression
    for p in os.listdir('.'):
        for f in os.listdir(p):
            epub.write(os.path.join(p, f)), compress_type=zipfile.ZIP_DEFLATED)
    epub.close()
```

That's it! **Remember to validate.**

Summary

The Python script in the [previous section](#) is just a first step in fully automating any kind of EPUB conversion. For the sake of brevity, it does not handle many common cases, such as arbitrarily nested paths, stylesheets, or embedded fonts. Ruby fans can look at `dbtoepub`, included in the DocBook XSL distribution, for a similar approach in that language.

Because EPUB is a relatively young format, many useful conversion paths still await creation. Fortunately, most types of structured markup, such as reStructuredText or Markdown, have pipelines that produce HTML or XHTML already; adapting those to produce EPUBs should be fairly straightforward, especially using the DocBook-to-EPUB Python or Ruby scripts as a guide.

Because EPUB is mostly ZIP and XHTML, there's little reason not to distribute documentation bundles as EPUB archives rather than as simple .zip files. Users with EPUB readers benefit from the additional metadata and automatic table of contents, but those without can simply treat the EPUB archive as a normal ZIP file and view the XHTML contents in a browser. Consider adding EPUB-generating code to any kind of documentation system, such as Javadoc or Perldoc. EPUB is designed for documentation at book length, so it's a perfect distribution format for the increasing number of online or in-progress programming books.

Downloadable resources

Description	Name	Size
Resources to build the EPUB in this tutorial	epub-raw-files.zip	8KB
DocBook to EPUB tools[1]	docbook-to-epub.zip	7KB

Note

1. This .zip file contains the sample DocBook XML file illustrated in the tutorial and a complete docbook2epub.py script. You must download lxml and the DocBook XSL separately; see the links in Resources.

Related topics

- Complete EPUB specifications
- XHTML 1.1 and DAISY: For more information on EPUB content formats, consult the XHTML 1.1 specification (currently a W3C Working Draft) and the DAISY Specification for the Digital Talking Book (DTBook).
- EpubCheck: Adobe EpubCheck is an invaluable tool for EPUB creation. Download and run it as a stand-alone program, a Web application, or as a library (requires Java version 1.5 or later).
- Dublin Core Metadata: For more on metadata terms available in Dublin Core, consult the DCMI Terms document and usage guide.
- DocBook XSL: Download the latest version of the stylesheets for processing DocBook into EPUB. The DocBook XSL package also includes a Ruby script for processing into a complete EPUB archive, similar to the Python script demonstrated in this tutorial.