EPUB Best Practices Guide

By the Adobe Digital Publishing Team

Version 1.0.3
August 2008

www.adobe.com/devnet/digitalpublishing

Introduction

Publishers, conversion houses, distributors, retailers and others are keenly interested in producing, distributing and selling eBooks and other commercial publications in portable, standard formats. PDF is broadly adopted as the open standard format for final-form page-oriented digital content. Recently, a new XML-based format, IDPF EPUB, has been standardized. EPUB complements PDF by focusing on reflow-centric publications that can be reliably adapted to different display sizes and user font preferences. EPUB adoption is rapidly increasing: rather than creating digital books in a variety of proprietary formats, publishers are choosing EPUB as their standard publication format for reflow-centric eBooks. Vendors, including Adobe, are implementing support for EPUB in their solutions.

This Guide provides overview information on EPUB and explains, at the file format level, how to produce portable, efficient EPUB content. It is targeted at developers within a publishing house who need to create software tools, scripts, and/or XML templates in order to produce EPUB in a reliable manner, from other XML formats or in-house tools. A basic understanding of XML technology and tools is assumed.

Adobe's InDesign CS3 software can export EPUB content directly. If you are using this tool, or another commercial software program with EPUB support, much of the information in this Guide may not be applicable. But, it still may be useful background to understand the trade-offs and considerations that software vendors have to make in creating EPUB content, and because EPUB is an XML-based format it is straightforward to create downstream workflows for further manipulating or reusing EPUB content created by such commercial programs.

Following the best practices in this Guide, you will produce EPUB files, compatible with the IDPF EPUB specifications, that will look great for end consumers, regardless of whether they are consuming your content on desktop PCs, mobile phones, or other devices.

You can read this white paper, which is published in EPUB format, by downloading the latest beta version Adobe Digital Editions at http://labs.adobe.com/technologies/digitaleditions . Feel free to e-mail this guide to anyone you think who would find this useful.

Intended Audience

As a guide for producing EPUB publications, this document is intended for a technical audience. The goal of this guide is to disseminate an understanding of the EPUB format, and in doing so promote the production of tools and procedures that will reliably create good EPUB documents. A good EPUB document is one that works well across the spectrum of compliant Reading Systems.

EPUB Overview

The EPUB file format was developed and is maintained by the International Digital Publishing Forum (IDPF, www.idpf.org), a nonprofit standards organization composed of commercial trade and academic publishers, software companies, consumer electronics manufacturers, and publishing and accessibility associations. The standard was developed with the participation of over 60 companies and organizations and was unanimously approved by IDPF members.

EPUB Specifications

The EPUB file format comprises three specifications: Open Container Format (OCF), Open Publication Structure (OPS), and Open Packaging Format (OPF). OCF is simply a zip-based standard used to encapsulate all of the pieces of a digital publication into a single file. OPS describes the digital publication's markup or content (words on the page), and OPF provides the navigation, packaging, metadata, and table of contents (how the pages relate to one another). The extension that denotes this file format is .epub, just as .pdf is the file extension that denotes PDF documents. EPUB is the successor to the Open eBook Publication Structure (OEB) specification. Creation of EPUB from OEB is a straight-forward process.

Adobe Digital Editions and EPUB

Adobe Digital Editions users can read their digital publications in either PDF or EPUB format. They can also organize and read PDF and EPUB files on all of the mobile devices that Adobe Digital Editions supports.

The advantages of EPUB and PDF

Adobe Digital Editions supports EPUB and PDF file formats because of the advantages that both offer. PDF represents a fixed-page view and gives you complete control over page layout and presentation. The reader consumes content exactly as the publisher intended. EPUB allows the publication text to reflow according to screen size, enabling the publisher to distribute and the reader to consume digital publications on a variety of screen sizes. Adobe Digital Editions and the mobile devices that it supports can read both PDF and EPUB digital publications.

An open standard for digital publications

EPUB is developed and maintained by the IDPF, a nonprofit standards organization.

EPUB specifications are based on open standards such as XML, XHTML, CSS, Unicode, DTBook, OASIS Open Document Format, and others. There is no fee associated with the creation, distribution, or consumption of the EPUB file format.

Creating EPUB files in Adobe InDesign® CS3

Adobe InDesign CS3 software allows you to export EPUB files. Beautiful and rich commercial content can be created directly from InDesign CS3, which includes advanced EPUB options for text styling, font embedding, and CSS styling as well as direct export of images, objects, and embedded Adobe Flash® multimedia files within the EPUB file.

Creating EPUB files with Content Conversion Houses

There are a number of content conversion companies who provide production services and can produce well-formed EPUB files. This guide will aid publishers and content conversion houses in producing the best possible EPUB files.

EPUB and multimedia

The EPUB specifications allow the integration of multimedia components to create engaging experiences in digital publications. Those publications can be read on a PC or a variety of mobile devices both online and offline. Publishers can include interactive games, quizzes, charts, tables, and media such as music and video into their EPUB files. The specifications also provide a full set of fallback mechanisms so that, regardless of the reading system, the reader experiences the publication in a way that is tuned to the environment.

Tools supporting EPUB and validation

The EPUB format is open and patent-unencumbered. Tools and applications are available from a variety of companies and organizations. Of particular note is an EPUB validation tool, developed in part by Adobe. The tool is available as an open source project. The purpose of the tool is to give you a way to check EPUB based content, ensuring that your EPUB files are compliant with IDPF specifications. Standard, validated EPUB files are much less likely to have problems in today and future eBook hardware and software. The EPUB validation tool is located at http://code.google.com/p/epubcheck . More on this validation tool below and how to best use it.

Accessibility, EPUB and NIMAS Conformance

The EPUB specifications allow markup in two vocabularies: XHTML and DTBook. DTBook is a National Information Standards Organization (NISO) standard that defines the format and content of an electronic file comprising a digital talking book (DTB). DTBs are designed to make print material accessible and navigable for sight-impaired and print-disabled persons.

K-12 publishers can create EPUB digital books using the DTBook vocabulary, with additional required information, to create a marketable commercial digital book file that also conforms to requirements of the National Instructional Materials Accessibility Standard (NIMAS).

EPUB and digital rights management

For commercial publishers, retailers, and distributors that wish to distribute and sell EPUB digital publications with copy protection, Adobe offers DRM services to protect EPUB and PDF digital books.

Consumer benefits of EPUB

Unencrypted EPUB files can be opened and read on any hardware or software that implements the standard. For unencrypted digital publications, this means that readers enjoy full interoperability between hardware and software. EPUB also allows publishers to easily produce content for sale, dramatically increasing the number and quality of digital publications for the consumer.

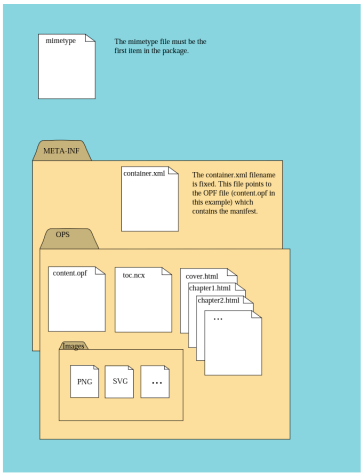
OCF 1.0

The OCF specification gives you a means to wrap up a multi-file EPUB document so that it looks and behaves as if it is a single document. The packaging of the files is straight forward but must follow a specific set of rules. Those rules and the technical details surrounding them can, of course, be found on the IDPF web site at:

http://www.idpf.org/ocf/ocf1.0/index.htm

OCF is a general purpose packaging standard, but our discussion of it here will be within the context of packaging EPUB documents.

Looking inside the package of any well structured OCF document you will find they have a few common characteristics.



You'll see that there is a mimetype file. The mimetype file identifies the content type within the package. It must be the first file in the archive and needs to have the EPUB mimetype as plain text within the file:

```
application/epub+zip
```

The mimetype file must be stored (uncompressed) in the package.

You'll also find that the document has a `META-INF` folder, that folder's name is fixed and that folder contains the 'container.xml' and possibly an 'encryption.xml' file.

Of course in any EPUB you're also going to find the content, typically this is contained in an OEBPS or OPS folder, the name isn't the important thing, it's that you have a folder that contains the rest of the document. Keeping your documents organized in the package is helpful at times especially when you're developing a system for generating EPUBs and tend to open a lot of EPUB documents to debug their content.

Encryption

The OCF v1.0 specification provides for encryption of the contents on an individual file level. This can be useful in mixing protected and unprotected content (i.e. encrypting the fonts used in a publication but not encrypting the rest of the contents.)

The approach for encryption is from the 'XML Encryption Syntax and Processing' (http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/).

When encrypting content in the OCF container you need to provide the Reading System with information on what has been encrypted and how it has been encrypted. This information is kept in the encryption.xml file which you would place in the META-INF folder. If any content is encrypted the encryption.xml file is required.

Within the root element of the encryption file (`<encrypiton>`) you will have `<EncryptedKey>` elements to identify keys for decrypting the files and you will have `<EncryptedData>` elements that will provide the URI for the encrypted file.

Below is the encryption file example from the IDPF web site (from the OCF 1.0 specification.)

As the site explains, the encryption document can have both `<EncryptedKey>` elements and `<EncryptedData>` elements. The `EncryptedKey` , in this example contains the key for the AES encrypted content. Since the key is in the document, the key itself has been encrypted with RSA encryption. That's why you see an `EncryptionMethod` and `CipherData` inside the `EncryptionKey` , those items are associated with the RSA encryption of the cipher key.

```
<encryption
xmlns
="urn:oasis:names:tc:opendocument:xmlns:container"
xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<enc:EncryptedKey Id="EK">
  <enc:EncryptionMethod
  Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
  <ds:KeyInfo>
    <ds:KeyName>John
    Smith</ds:KeyName>
  </ds:KeyInfo>
  <enc:CipherData>
    <enc:CipherValue>xyzabc</enc:CipherValue>
  </enc:CipherData>
</enc:EncryptedKey>
<enc:EncryptedData Id="ED1">
  <enc:EncryptionMethod
  Algorithm="http://www.w3.org/2001/04/xmlenc#kw-
  aes128"/>
  <ds:KeyInfo>
    <ds:RetrievalMethod URI="#EK"
    Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
  </ds:KeyInfo>
  <enc:CipherData>
    <enc:CipherReference
    URI="image.jpeg"/>
  </enc:CipherData>
</enc:EncryptedData>
</encryption>
```

It's worth noting that OCF encrypts individual files independently, trading off some security for improved performance, by allowing the container contents to be incrementally decrypted.

XML Encryption specifies a process for encrypting arbitrary data and representing the result in XML. Even though an OCF container will contain non-XML data, XML Encryption can be used to encrypt all data in an OCF container. OCF encryption supports only encryption of whole files.

Of course the encrypted data replaces unencrypted data in an OCF container. For example, if an image named "photo.jpeg" is encrypted, you would put the encrypted stream in the package in the location where the unencrypted stream would have been. The following files are never encrypted (regardless of whether default or specific encryption is requested):

- mimetype
- META-INF/container.xml
- META-INF/manifest.xml
- META-INF/metadata.xml
- META-INF/signatures.xml
- META-INF/encryption.xml
- META-INF/rights.xml
- OEBPS rootfile (the OEBPS Package file)

Compression

OCF supports files being stored without compression and being stored with 'Flate' compression. Fortunately, this is the default compression for most zip archiving tools and so there's many tools that support the OCF package even if they don't recognize the .epub file extension.

You'll need to add the mimetype to the package first, making sure to choose whatever options you need so that the mimetype is stored without compression and without the extra field in it's ZIP header. If you do that you should end up with a zip package that has the bytes "PK" at the beginning, "mimetype" at byte 30, and "application/epub+zip" and byte 38.

If those items aren't at the right byte locations, it's often because of the 'extra field' in the zip header. The result of which is that epubcheck will claim that there is no mimetype, and yet the mimetype will appear to be in the package.

Organization: Open Packaging Format (OPF) version 2.0

The OPF version 2.0 specification is available on the IDPF web site, at:

http://www.idpf.org/2007/opf/OPF_2.0_final_spec.html

The OPF file

While OCF describes the overall packaging of the EPUB in a single file format, it provides a pointer (in the container.xml) to the root file of the document structure. That root file is described in the OPF specification. The OPF root file contains publication level metadata, references all of the components of the publication, gives a reading order for the contents, and gives an extension/fallback mechanism for files that are not core content types. The OPF file is typically named content.opf, but this is not required - only that it be correctly pointed to from the container.xml file

Structure of the OPF file.

The package element has two required attributes, the `version`, and the `unique-identifier`. You will need to reference one of the identifiers from the metadata. (While you only need to have one identifier, you can have multiple identifiers, for instances a record number for the book in a tracking system and the ISBN number for a book.) The `unique-identifier` should, as the name implies, indicate an `<identifier>` that is unique to this document.

```
<package
xmlns="http://www.idpf.org/2007/opf"
unique-identifier="bookid"
version="2.0">
...
</package>
```

Within that package element you'll need a <metadata>, a <manifest> and a <spine>, you may also have a <guide>.

Metadata

The OPF uses the Dublin Core metadata elements set without alteration or addition, and the specification defers to the official description, which can be found here:

http://dublincore.org/documents/2004/12/20/dces/

Please note that the preceding version of the OPS spec, OEBPS 1.2, erroneously listed the metadata element-names with the first letter capitalized, e.g. Title. This was incorrect and is not supported in OPS 2.0 unless the package is explicitly declared as a legacy package by omitting the package version. Do note, though, that this usage is deprecated and authors are strongly urged to use the correctly-case metadata element names.

The package has a metadata element which is intended to provide a full description of the publication, not only to the reader but to book sellers, online stores, and systems in between, so it's a good idea to provide all the information that is available. There are three items that your metadata must have, a `<title>` element, one or more <identifier> elements, and a `<language>` element.

```
<metadata
xmlns:dc="http://purl.org/dc/elements/1.1/">
  <dc:title>A Book</dc:title>
  <dc:identifier
  id="bookid">urn:uuid:719ca56c-416e-
  fe1c-cf57-
  e4fa79bd5532</dc:identifier>
  <dc:language>en</dc:language>
    ...
</metadata>
```

The rest of the Dublin Core metadata element set is optional, but you'll find many of the items listed helpful in providing a full metadata set for your books.

Manifest

The manifest should reference all of the files that make up the content of the document as well as any stylesheets, fonts and images that go along with the content. For each `<item>` there are required `href`, `id`, and `media-type` attributes. There can also be a fallback attribute. The fallback attribute is required for any XML content document that is not a core data type.

```
<manifest>
  <item id="ncx" href="toc.ncx"
  media-type="application/x-dtbncx
  +xml"/>
  <item id="untitled-1"
  href="Untitled-1.xhtml" media-
  type="application/xhtml+xml"/>
  <item id="css" href="template.css"
  media-type="text/css"/>
</manifest>
```

Spine

The spine is a reading order listing of all the content documents. This list should not include the stylesheets, fonts, OPF file, or images. (The images are generally referenced from the XML documents. You should note that the reference to the NCX file is in the `toc` attribute, and not in a child element.

```
<spine toc="ncx">
<itemref idref="untitled-1"/>
</spine>
```

Guide

The package can have an optional `<guide>` element. The structural components of the books are listed in reference elements contained within the guide element. These components could refer to the table of contents, list of illustrations, foreword, bibliography, and many other standard parts of the book. Adobe Digital Editions makes no use of the guide currently.

The NCX file

The NCX file, which is referenced above in the `<spine>` element of the OPF file (above) provides the Reading System with detailed navigation information. Where the `spine` is a reading-order listing of the XML files that make up the content of the document, the NCX is multileveled and provides links to the chapter documents as well as links to sections within those chapters.

It's important to note that while it supports fragment identifiers (the string after the hash after the URI), they should not be used in the top level `<navPoint>` elements. This may force reading systems to open each chapter to build it's navigation tree, which could pose performance problems on slower devices. Using fragment identifiers on the second (and lower levels) doesn't pose the same issue because the reading systems can defer processing those levels until they are needed.

```
<!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="03318f50-ac37-
a161-8699-70bbf0f47fae"/>
<meta name="dtb:depth" content="1"/>
<meta name="dtb:totalPageCount"
content="0"/>
<meta name="dtb:maxPageNumber"
content="0"/>
</head>
<docTitle>
<text>A Book</text>
</docTitle>
<navMap>
<navPoint id="navpoint"
playOrder="1">
<navLabel>
<text>Untitled-1</text>
</navLabel>
<content src="Untitled-1.xhtml"/>
</navPoint>
</navMap>
</ncx>
```

Content

Body content (XHTML)

Use Chapter size chunks (less than 300k in size.)

As noted above the EPUB format supports both XHTML and DTBook content within the EPUB package. The content of the document should be broken up into multiple files. Having a single XHTML document that's the entire contents of a novel may be technically valid, but that would also mean that the entire document would need to be loaded into memory when the first page gets rendered or when the user opens the table of contents. It's much better for reader performance, navigation and usability to split the document into chapter or even section size chunks. Typically you'll want to treat chapters as separate chunks, in some cases, when the chapters are very long, you'll want to break them up further. Of course the start of each new chunk will start at the top of a rendered page, so you'll want to split the chunks with that in mind.

Note that Adobe Digital Editions has the following limitations when running on a mobile device. If these limits are not adhered to, EPUB files will not work on supported mobile devices including the SONY® Reader Digital Book.
Image Size: 10MB uncompressed.
XHTML/DTBook file size: 300k uncompressed/100k compressed.

The limits shown above are per asset within the document. Since your books will have many 'chunks' or chapter files, the full text can be much longer than the 300k limit. The limit is only a limit on the individual pieces.

Use block-level elements

Good document structure is based on using semantically correct markup. If you have a heading use the heading elements. If you have a paragraph, use a paragraph element. If you have a list, use a list item element. If you have a chapter or section, naturally the heading will be marked with the appropriate level heading, but it is also be very useful to wrap the chapter or heading in a `div` element. Having chapters and sections wrapped in `div` elements will also help when you go to style the document with `CSS` .

Use `<br />` element only when it's appropriate (poems, addresses, etc.)

The `br` element is for introducing line breaks where those breaks are part of the content. The breaks within verses of a poem are part of the content and so that use is helpful. Other use of the `br` element should be avoided. Paragraphs will already have breaks between them because of the paragraph elements and you'll be able to control the space around them with `CSS` styling.

Stylesheets

CSS styling

Since an EPUB reader will use your CSS stylesheets in rendering all the pages of the book, the stylesheet represents an opportunity to set your works apart from the others, but can also be a pain point if constructed in such a way that causes the Reading System (e.g. Digital Editions) an unwarranted amount of processing to render each page.

Use external style sheets

Using external stylesheets makes it easier to create and maintain a pleasing uniform look across all the chunks within the book. Of course if you are consistent in the naming and use of classes then you can create a single set of stylesheets that you use across a whole class of EPUB documents.

Use spacing that looks more like a book

There's default values for margins and padding in CSS, and they're not the sort of thing you want in a book, so get rid of them all by setting a new set of margins and padding for all of the block-level elements you plan to use in your documents, something like this:

```
body, div, p, h1, h2, h3, h4
{ margin: 0; padding: 0; }
```

Then follow that rule with rules that add the spacing you want.

Use simple selectors

Don't get carried away with the selectors. You can use class selectors to avoid complex selectors and the rendering of the document will go much smoother.

Bad:

```
div > p > span:first-child {  font-size: 1.5em }
```

Better:

```
span.highlight {  font-size: 1.5em }
```

Keep unused styles to a minimum

Extra, unused styling in the stylesheets just add processing overhead to the pages without adding any value. Remove those extra styles and you'll have documents that behave better on any device. This is especially true on smaller, lower-powered devices. In general, trying to make the book as simple as possible will really help with slower devices. A book may read just fine on a high-end desktop but be sluggish when viewed on a PDA or phone.

Avoid dynamic styling

The CSS dynamic pseudo-class selectors (`:active` , `:hover` , and `:focus`) cause the entire page to be redrawn, and therefore should not be used.

Use relative sizes

Use em's or percentage values for any item that should grow or shrink when the user chooses a larger or smaller font size. The adjustment can, and at times should affect not only the size of the text but also images.

Raster Images

Raster images present a bit of a challenge. The EPUB format is designed to work well on many different types of screens and so it is intended to scale up or down really well. However when you add images to the document, you'll need to choose one specific resolution for each of those images. For images that represent the cover or other full page illustration, you'll want a sizable image, so that the art looks good, but don't make it too big or performance on mobile devices will suffer. Keeping key full-page illustrations to no larger than 1200 x 1600 and other full-page illustrations to 600x800 will help manage file size. Images that don't occupy the entire view should be proportionally smaller in resolution.

Vector Art

SVG being a vector art format will support all views equally well, and therefore will give you sharp looking graphics even on the largest monitors while keeping file size in check. Of course to produce books with SVG you would need a workflow and tools that support it. Note that the SVG supported by the EPUB specifications is a subset of SVG 1.1, and that animation in the SVG is not allowed.

Character sets and Fonts

The character set that Adobe Digital Editions supports is enumerated in the PDF Reference, fifth edition (http://www.adobe.com/devnet/pdf/pdfs/PDFReference16.pdf). See Appendix D "Character Sets and Encodings", table D.1. We also support characters from the table D.3 (these are mostly symbols) as long as there is a corresponding Unicode character value for them. If you want glyphs outside those sets you'll need to use embedded fonts.

If you do use embedded fonts, then any unicode character supported by the fonts will work.

Of course you may choose to use embedded fonts for various other reasons, such as setting your titles apart or if you are unhappy with the look of the generic fonts.

To embed fonts in a document, you need to include the font files, and you need to identify those font files in the CSS stylesheet. Below are two of the fonts used in this document, as they appear in the CSS.

```
@font-face {
font-family: “Myriad Pro”;
font-style: normal;
font-weight: normal;
src:url(Fonts/MyriadPro-Regular.otf);
}
@font-face {
font-family: “Myriad Pro”;
font-style: normal;
font-weight: bold;
src:url(Fonts/MyriadPro-Bold.otf);
}
```

Cover Page and Illustrations

Use SVG and the viewBox attribute for the cover page and for illustrations.

As the cover represents the book in many different places, it’s a worthwhile effort to make sure your covers look good, not only in a full screen view on a PC, but also in thumbnail views in a bookshelf and in your bookstore as well as on the screens of mobile devices.

The best solutions for cover images is to do them in SVG and use the viewBox attribute to keep the cover image in view and undistorted. Here’s a bit of HTML code showing an inline SVG:

```
<body class=”cover”>
<svg version=”1.1”
xmlns=”http://www.w3.org/2000/svg”
xmlns:xlink=”http://www.w3.org/1999/xlink”
width=”100%”
height=”100%”
viewBox=”0 0 500 656” >
<rect x=”0” y=”0” fill=”white”
width=”500” height=”656”/>
<image width=”500” height=”408”
xlink:href=”images/XML_Boy.png”
transform=”translate(0 135)” />
<rect x=”0” y=”0” fill=”#A10000”
width=”500” height=”15”/>
<rect x=”0” y=”557” fill=”#A10000”
width=”500” height=”98”/>
<rect x=”0” y=”35” fill=”#D6D6D6”
width=”500” height=”75”/>
<rect x=”0” y=”0” fill=”#FFFF99”
width=”55” height=”656”/>
<text transform=”matrix(0.5 0 0 1 70
97)”
font-family=”’AGaramondPro-Bold’,
serif”
font-size=”72”>MAD SCIENCE FOR
KIDS</text>
<text transform=”matrix(0 1.5 -1 0 15
50)”
fill=”white”
font-family=”’AGaramondPro-Bold’,
serif”
font-style=”bold” font-
size=”36”>BUILD COOL STUFF</text>
<text transform=”translate(74 610)”
fill=”white” font-family=”’Helvetica-
Bold’”
font-size=”24”>Paul Norton </text>
<text transform=”translate(225 610)”
fill=”white” font-family=”’Helvetica-
Bold’”
font-size=”18”>illustrated by </text>
<text transform=”translate(344 610)”
fill=”white” font-family=”’Helvetica-
Bold’”
font-size=”22”>Dawn Norton</text>
</svg>
</body>
```

Of course you could have the cover in an image and just wrap the image in the SVG, which would simplify the SVG but wouldn’t scale as well as the SVG. You can create particularly good looking covers by overlaying an image with the actual text of the title, author etc. Since the text is rendered independent of the screen size it gives an excellent effect,

Adobe Extensions

Page-map

XHTML-based ePub documents (as defined by OPF/OPS 2.0) have two distinct, but closely-related shortcomings:

1. There is no inherent linear navigation indicator which could be used for the same purpose that page number is used in the printed document world.
2. There is no way for an eBook to incorporate page number information for the printed edition of the same book. Note that page numbers sometimes can be Roman numerals or a pair of numbers.

A related problem is being able to indicate a page range when printing a section of a book.

Page Map resource

Page Map is a resource which can be included in the EPUB file and referenced in the OPF manifest. The proposed media type for it is application/oebps-page-map+xml. Id for the page map must be referenced by the spine element using page-map attribute.

Page Map syntax

The page-map file has a simple syntax: root element name page-map that contains a list of page elements. Each page element must have a name attribute that indicates a page name and href attribute that indicates a reference to the first piece of content on that page. XPointer can be used to allow referencing down to individual characters. The name attribute is often a number, but can be any string. For the cover, you may want to give it an empty string, so that no "page number" or name shows up next to the cover.

In addition, a metadata element is allowed as a first child of the page-map element. This element can contain metadata items that are specific for a printed edition for which the page map is recorded (publisher, date and ISBN should be given if available).

Example:

```
<page-map
xmlns="http://www.idpf.org/2007/opf">
  <page name=""
  href="OPS/cover.xhtml"/>
  <page name="i"
  href="OPS/preface.xhtml"/>
  <page name="ii"
  href="OPS/preface.xhtml#pg_ii"/>
  <page name="iii"
  href="OPS/preface.xhtml#pg_iii"/>
  <page name="1"
  href="OPS/chapter1.xhtml"/>
  <page name="2"
  href="OPS/chapter1.xhtml#pg_1"/>
  <page name="3"
  href="OPS/chapter1.xhtml#pg_2"/>
  <page name="4"
  href="OPS/chapter2.xhtml"/>
  <page name="5"
  href="OPS/chapter2.xhtml#pg_1"/>
  <page name="6"
  href="OPS/chapter2.xhtml#pg_2"/>
</page-map>
```

Synthetic page names

When page map is not available in the document, Adobe Digital Editions will synthesize a page-map based on the document content. The approach used is the following:

1. Determine a compressed byte length of each resource which is referenced in the spine, subtracting any known encryption overhead (IV size)
2. Assume that there is a page for each 1024 bytes in each resource, rounding up to the nearest whole number of pages for each resource
3. To map page breaks into a resource, use the number of pages for the resource as determined in step 2, count the number of Unicode characters in the resource; distribute synthetic page breaks in the resource evenly between the characters by dividing the number of characters by the number of pages; if the number of characters don't divide evenly among the pages, round the number of characters per page up and let the last "page" contain less characters than the rest.

Layout Template

Adobe Digital Editions viewer implements several layout extensions to EPUB format. These include:

- Page Masters – specify XSL:FO page masters to add headers, footers and sidebars as well as multi column layout.
- Dynamic Page Master Selection – choose the right page master based on the environment.
- Dynamic Styling – style the document based on the environment (viewing area dimensions, default font size, device resolution, etc.)

These extensions are typically used together and can be all packaged as an additional stylesheet, called XML Page Template (XPGT). It can be referenced like any other stylesheet directly from an XHTML file:

```
<html
xmlns="http://www.w3.org/1999/xhtml">
<head>
  ...
  <link rel="stylesheet"
  type="application/adobe-page-
  template+xml"
  href="template.xpgt"/>
</head>
<body>
  ...
</body>
</html>
```

Alternatively, an XPGT file can be pulled in from a CSS stylesheet with import statement:

```
  @import url(template.xpgt);
```

The XPGT file is also declared in the OPF manifest and assigned media type "`application/adobe-page-template+xml`". As you might have guessed from the media type, XPGT is using XML rather than CSS-like syntax. Here is how XPGT file is structured:

```
<ade:template
xmlns="http://www.w3.org/1999/xhtml"
xmlns:ade="http://ns.adobe.com/2006/ade"
xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
  <!-- list of page master
  definitions -->
  <!-- page master selection -->
  </fo:layout-master-set>
  <ade:style>
  <!-- dynamic styling rules -->
  </ade:style>
</ade:template>
```

Page Master Definitions

This section contains one or more page master definitions (they are called simple page masters in XSL:FO). A page master defines a partitioning of the page's surface space into a set of regions. Each region is an area which can be occupied by a flow. A flow is a sequence of flowable content – paragraphs, lists, images, tables, etc. A sample page master is shown below:

```
<fo:simple-page-master master-
name="two_column_head">
<fo:region-before extent="6em"/>
  <fo:region-body column-count="2"
  column-gap="10pt"
  margin-bottom="1em" margin-
  top="6.2em"
  margin-left="0.3em" margin-
  right="0.3em"/>
</fo:simple-page-master>
```

The mark-up above defines a page master named "two_column_head" that splits the page into a 6em-high header region and a body region. The body region has two columns in it. There is 10pt gap between columns and there are margins around the body region. Note that each region is positioned independently in XSL:FO, therefore the top margin of the body region is enlarged to avoid an overlap with the header region. By default region names in the example above are "xsl-region-before" and "xsl-region-body". Different names can be given using region-name attribute.

Page Master Selection

The next section specifies how to build a sequence of pages in the document using previously defined page masters (possibly different masters for different pages). Again, XSL:FO mark-up is used. Several extensions are available to achieve dynamic effects.

```
<fo:page-sequence-master>
<fo:repeatable-page-master-
alternatives>
  <fo:conditional-page-master-
  reference
  master-reference="two_column_head"
  page-position="first"
  ade:min-page-width="50em"/>
  <fo:conditional-page-master-
  reference
  master-reference="two_column"
  ade:min-page-width="50em"/>
  <fo:conditional-page-master-
  reference
  master-reference="single_column"/>
</fo:repeatable-page-master-
alternatives>
</fo:page-sequence-master>
```

The meaning of the above mark-up snippet is: "repeatedly choose the first acceptable page master from the list". What is "acceptable" is determined by the conditions on the fo:conditional-page-master-reference element attributes. The attribute page-position="first" specifies that this page master is only acceptable for the first page. The attribute ade:min-page-width specifies minimal acceptable width of the page for a page master. It is also possible to specify ade:condition attribute and place any general condition there as an XPath expression (in curly braces).

Dynamic Styling

The final section describes styling rules that depend on the environment (dynamic styling). These rules work together with rules CSS stylesheets. The rules in the page template have higher priority (specificity in CSS terms). Therefore CSS stylesheets should be used to specify environment-neutral styling of the document and page template styling should be used for environment-specific styling.

Just like CSS rules, page template styling rules have a selector which determines a set of applicable elements and a set of properties that the rule assigns to the elements. Selector is specified using selector attribute using regular CSS syntax. Properties are specified using attributes with matching names. In addition, styling rule can contain a condition which determines if the rule should be applied given a particular environment. Here is an example:

```
<ade:styling-rule selector=".ibox"
condition="{ade:page-width() &gt;
22*ade:default-font-size()}"
float="left" width="70%" margin-
top="0pt"
margin-right="0.5em" margin-
bottom="0pt" margin-left="0pt"/>
```

This rule applies to all elements with class "ibox". It sets a number of properties (float, width, margins) of these elements. It is only applied when page width is greater than 22 times default font size.

Conditions are specified using XPath expressions in curly braces. Here is a list of XPath extension functions to query the environment:

`ade:page-width()` The width of the page in CSS pixels

`ade:page-height()` The height of the page in CSS pixels

`ade:page-aspect()` Page width divided by page height

`ade:default-font-size()` Default font size in CSS pixels

`ade:device-is-color()` True if the device supports color

`ade:resolution()` Size of the inch in CSS pixels

It is also possible to assign values of the properties based on the environment by specifying property value as an XPath expression in curly braces:

```
<ade:styling-rule selector=".top"
condition="{ade:page-width() &lt;
36*ade:default-font-size()}"
font-size="{ade:page-width() div
10}"/>
```

This styling rule assigns font size of the elements with class "top" to 1/10th of the page width when page width becomes less than 36 times default font size.

Assigning Content to Page Master Regions

By default all the content of an XHTML file goes into "xsl-region-body" region. It is possible to assign elements to different regions by specifying display and adobe-region properties. It should be done with page template styling (it is also possible to do with CSS, but since page template must be specified anyway, it makes sense to put all page-template related styling in page template itself).

```
<ade:styling-rule selector=".top"
display="adobe-other-region"
adobe-region="xsl-region-before"/>
```

" `adobe-other-region` " is a special value for display property that makes an element to display as a block in the different region. The region name is specified by the adobe-region property. If such region does not exist, an element is simply displayed as a block.

EPUB Documents without Page Template

There are always going to be many documents that do not choose to incorporate a specific page template. Indeed it is a goal that documents that do not wish to declare custom intentions with respect to page layout should not need to incorporate such markup. Obviously, a Reading System that supports the page template extension must be able to process such documents.

One strategy would be to render such documents in a single column, no matter how large the viewing are is and how small the font size is. This would mean that such documents would look quite ugly. So if no page template is specified Adobe Digital Editions 1.0 automatically switches to 2 columns when width is about 61.5em and to three column layout when width is 93em. Column gap is about 1.5em. These numbers may change in future.

Relationship to oeb-* CSS properties

OPS 2.0 specifies its own extensions to CSS to set the number of columns in the form of oeb-prefixed properties. Currently Digital Editions does not support these properties, however our goal is eventually to support complete OPS 2.0 spec.

Relationship to CSS3 features

CSS specification is evolving beyond CSS2.1 in the shape of multiple CSS3 modules. Several of CSS3 modules aim to the solve the same problems as page template: adding headers and footers, multi-column layout and dynamic styling. All of these modules are still being designed and cannot be relied upon. Since CSS group chose to ignore a lot of previous work in this area (e.g. inventing its own expression syntax for media queries instead of using XPath, designing its own incompatible replacement of page master instead of building on top of XSL:FO), the amount of the implementation work that would be required to achieve parity even with Digital Editions 1.0 is huge. This does not preclude implementing CSS3 syntax for these features if they become stable and gain some acceptance, but it makes too expensive for us to experiment with them at this time.

Practices to Avoid

Using
 to format paragraphs

The `br` element is for introducing line breaks where those breaks are part of the content. An example of the appropriate use of `<br />` would be the breaks within verses of a poem, as those are part of the content. Other use of the `br` element, like to end a paragraph or to enforce some styling rule should be avoided. Paragraphs will already have breaks between them because of the paragraph elements and you'll be able to control the space around them with `CSS` styling.

One XHTML document for an entire book

Really big files with plain ASCII text provide a very poor user experience in comparison to a book. The editors for .txt files don't remember where you were in a book, don't provide navigation, don't provide a consistent view of the text, won't let you create bookmarks, and make it all but impossible for a book group to discuss what happened on page 688.

When you open an EPUB, there's much that a reading application can do for you, it will layout pages of text, find book marks and hyperlinks in the text (like, say if you are using footnotes in InDesign) and so the book isn't really treated so much as one long string of plain text as it is the content of a book that needs to be paginated on the fly and managed. Now, how much goes through this process of pagination depends on the size of the chunks within the EPUB. If you have one really long chunk (XHTML file) that is a 100+ chapter book, then the application will have to paginate that whole thing at once.

The issue really isn't as noticeable on a desktop PC, but on a cell phone or other handheld device a single book all as one XHTML file isn't a good use of the resources on the device.

For Adobe Digital Editions, you will need to keep chapters under 300k in size when uncompressed and under 100k when compressed. Images must not be larger than 10MB when uncompressed. If these limits are not adhered to, EPUB files will not work on supported mobile devices including the SONY® Reader Digital Book.

Hidden content / Interactive styling

Hidden content and interactive styling can be used to create some very interesting effects on the desktop, but cause problems for the EPUB documents both on the desktop and on smaller devices. The obvious situation is where a footnote could be hidden, or actually just shrunk to be a single pixel tall, until a certain item is "hovered" with the mouse. In this troublesome scenario, the text would then be changed to a readable size, and therefore would seem to 'pop-up' like a tool-tip. You should avoid the :hover pseudo-class in any case, as it implies a certain amount of interactivity that may not exist in all cases (consider screens based on e-Ink technology, where the screen refresh is likely to be a single frame or two frames a second.) Such dynamic styling can have other unexpected side-effects, such as hiding the results of a search and therefore frustrating the user.

Fragment Identifiers for chapter level links

When you look at the markup for the TOC in the NCX file, you'll find that it has a hierarchy of navPoint elements and each navPoint element has a content element that identifies the location of the content in a src attribute. These links, just like the href links within an XHTML document can point to a document at the file level ('Chapter1.xml') or it can add a fragment identifier to indicate a location within the file ('Chapter1.xml#Section1')

When the Table of Contents for a book is displayed, the system will also locate the destination for each link in the top level of the navMap.

SWF movies without fallback content

When including SWF movies for desktop viewing, it's important to realize that they won't appear on devices

that don't support animation. This doesn't mean that you cannot use SWF movies in your documents but you would need to provide static content as a fallback to the SWF content.

The <object> tag that is used to include a movie in an EPUB document is conditional by it's very nature. The item in the data attribute will be played if the current environment supports it, otherwise the content within the object tag will display.

Tools for Building and validating EPUB documents

EPUB Checker

It is important to make sure that EPUB content satisfies the standards. This is especially important to make sure that eBooks work well on mobile devices where adding special-casing to "fix" broken content might be simply too expensive. The best way to make sure that the content is standard is to develop a tool that can validate EPUB files.

The need for such tool was for a long time recognized by IDPF members. As we were developing EPUB standard, we did the work on validating EPUB files. Now this tool (named 'epubcheck') is available as an Open Source project. It is not complete (there are still many checks that we can do), but it is already fairly mature and extremely handy. If you author EPUB files, you should consider running this tool on your content regularly. Standard content is much less likely to have problems in today and future eBook readers and any problems with fully-compliant eBooks are much more likely to get serious attention of the developers.

The epubcheck tool, instructions, discussion, and development are available here:

http://code.google.com/p/epubcheck/

DocBook XSL

At the time of the writing of this document, there were a set of stylesheets, being developed for inclusion in the DocBook XSL project. Those stylesheets were functional and undergoing user testing. The plan is for them to be included with an upcoming distribution of DocBook XSL.

http://docbook.sourceforge.net

OEB 1.2 to EPUB conversion

It certainly is possible to convert OEB 1.2 to EPUB. Here are the key elements to bear in mind when converting OEB 1.2 content to OPS and OPF.

The content needs to be converted to XHTML if it is not already well-formed XML.

EPUB does not allow for documents that are not well-formed XML, so you'll want to convert the content documents. As the documents aren't necessarily well-formed, you may need to look at HTML Tidy, Beautiful Soup, or another tool that can generate an XML DOM from an HTML document.

Remove the deprecated items.

While not absolutely required, you should remove the deprecated '`tours`' element and move the metdata out of the '`dc-metadata`' element so that the individual items are direct children of the '`metadata`' element. The deprecated items won't be useful for any reading system that supports EPUB, and so these items aren't needed.

Change the metadata elements.

Metadata tags should be lower-cased (unless the package is left without a version). This would also be a good opportunity to review the metadata content and make sure it's correct and up-to-date.

Create an NCX file.

Generate an NCX file (based on the `spine`, or if it's available based on the document identified as the `TOC` in the `guide` element.)

Adobe is working on a set of scripts to make this process as smooth as possible, but it is important to check on the structure of the document as well as it's syntax, as detailed elsewhere in the document.

Useful Links

IDPF

http://www.idpf.org/specs.htm

Open Publication Structure (OPS) 2.0 v1.0

http://www.idpf.org/2007/ops/OPS_2.0_final_spec.html

Open Packaging Format (OPF) 2.0 v1.0

http://www.idpf.org/2007/opf/OPF_2.0_final_spec.html

OEBPS Container Format (OCF) v1.0

http://www.idpf.org/ocf/ocf1.0/download/ocf10.htm

ANSI/NISO Z39.86 - 2005 Specifications for the Digital Talking Book, NCX part (NCX)

http://www.niso.org/standards/resources/Z39-86-2005.html#NCX

DCMI Metadata Terms 2006-12-18 (DC)

http://dublincore.org/documents/2006/12/18/dcmi-terms/

Digital Editions Developer Center

http://www.adobe.com/devnet/digitalpublishing/