

Maak thuis Uw eigen e-book in Uw vrije tijd zonder schroeven

Pieter Masereeuw

Maak thuis Uw eigen e-book in Uw vrije tijd zonder schroeven

Pieter Masereeuw

Table of Contents

1. Maak thuis Uw eigen e-book in Uw vrije tijd zonder schroeven	1
Voor we beginnen	1
Wat zit er in een ePub-bestand	1
De “echte” inhoud	2
Tekst	2
Noten	4
Afbeeldingen	4
Fonts	4
Administratieve bestanden	5
Het metadatabestand	5
Het TOC-bestand	8
De resterende lijm: <i>mimetype</i> en <i>META-INF/container.xml</i>	10
Het aanmaken van het ZIP-bestand	11
De epub-checker	12
Verwijzingen	12

List of Figures

1.1. Afbeelding 1: het maken van losse XHTML-bestanden	3
1.2. Afbeelding 2: <i>CSS-fragment met font-declaratie</i>	5
1.3. Afbeelding 3: de hoofdstructuur van het bestand <i>metadata.opf</i>	6
1.4. Afbeelding 4: de manifest- en spinesecties van <i>metadata.opf</i>	7
1.5. Afbeelding 5: afbeeldingen opnemen in <i>metadata.opf</i>	8
1.6. Afbeelding 6: <i>de eerste twee secties van bestand toc.ncx</i>	9
1.7. Afbeelding 7: geneste navigatiepunten in bestand <i>toc.ncx</i>	10
1.8. Afbeelding 8: <i>de mappenstructuur van een ePub-bestand</i>	11
1.9. Afbeelding 9: <i>de file container.xml</i>	11

Chapter 1. Maak thuis Uw eigen e-book in Uw vrije tijd zonder schroeven

Door Pieter Masereeuw

Dat e-books populair zijn, zal niemand ontgaan zijn. In dit artikel kunt u lezen hoe een e-bookbestand in elkaar zit, meer precies gezegd: hoe een ePub-bestand in elkaar zit. Het ePub-formaat is in ons land het gangbare formaat voor e-books, zeker sinds eind 2009 toen Bol.com en Sony onze markt bestormden.

Natuurlijk kunt u op zoek gaan naar een tool om uw bestanden naar ePub te converteren. De zoekmachine zal u daarbij graag op weg helpen. In dit artikel laten we echter zien hoe je zelf XML-bestanden naar ePub kunt converteren. De nadruk ligt hierbij op *zelf*, vandaar de op het bijna gelijknamige bulkboek van Drs. P. gebaseerde titel.

Dit artikel is een voortzetting van het overzichtsartikel over ePub in de <!ELEMENT van oktober 2009.

Een ePub-versie van deze tekst kunt u vinden op www.masereeuw.nl/epub-maken.epub.

Voor we beginnen

In bladen voor een algemeen publiek is het gebruikelijk om allerlei verontschuldigen te uiten zodra een tekst technisch wordt. Dat doen we hier niet. Deze tekst gaat over techniek. Geniet ervan, of zap snel door naar een ander artikel.

Technische teksten bevatten soms een overdaad aan leukigheid om de taaie kost pruimbaar te maken. Notoir dieptepunt zijn daarbij de boeken over Perl. Perl is (misschien wel daardoor) nooit mijn favoriete taal geworden, maar de gortdroge boeken van Michael Kay reken ik tot mijn dierbaarste bezit. Verwacht dus geen grapjes. Trouwens: bij Michael Kay is tussen de regels door genoeg kwaliteitshumor te vinden. Hier niet.

In dit artikel converteren we een XML-bestand naar een ePub-bestand. Invoerbestand is een aflevering van <!ELEMENT, geschreven conform de <!ELEMENT-DTD. Dit is een beperkte versie van de DocBook-DTD. We doen de conversie met XSLT 2.0.

Wat zit er in een ePub-bestand

Een ePub-bestand is een ZIP-bestand dat volgens een aantal eenvoudige, maar specifieke regels is samengesteld. Globaal gesproken bevat het twee soorten onderdelen:

- tekst, afbeeldingen en eventuele andere nuttigheden, zoals fonts;
- administratieve bestanden.

Beide soorten onderdelen komen in dit artikel aan bod.

Alle onderdelen zitten in een mappenstructuur. Bij het samenstellen van deze structuur heb je de volledige vrijheid, afgezien van het volgende:

- De eerste file binnen de ZIP-file is de file *mimetype*, met als inhoud *application/epub+zip*. Deze file mag niet worden gecomprimeerd.

- De ZIP-file bevat een map met de vaste naam *META-INF* en daarin een bestand met de vaste naam *container.xml*. Deze file bevat de informatie waarmee de e-reader of het leesprogramma de andere bestanden kan vinden.
- Een map met daarin de “echte” inhoud: teksten, afbeeldingen en dergelijke. De naam van deze map is vrij (strikt genomen hoeft je niet eens een map te gebruiken, al wordt het aanbevolen). Een vaak gebruikte naam voor de map met de bestanden is *OEBPS*.

Deze structuur is vastgelegd in het OEBPS container format (OCF), opgenomen binnen de ePub-standaard (OEBPS is een afkorting van Open eBook Publication Structure). Aan het eind van dit artikel, als alle onderdelen genoemd zijn, laat ik een voorbeeld zien van de mappenstructuur.

De “echte” inhoud

Tekst

De tekst van een ePub-publicatie is meestal gebaseerd op XHTML plus CSS. Eigenlijk wordt aangeraden om geen XHTML te gebruiken, maar de DTBook-DTD. DTBook – Digital Talking Book – lijkt wel wat op XHTML, maar heeft meer boekachtige voorzieningen, zoals paginanummers, noten, frontmatter, backmatter, kolommen, sidebars en een beter afgedwongen documentstructuur. Tevens bevat DTBook SMIL-uitbreidingen voor geluid en spraak – al dan niet synthetisch.

Ik heb wel eens met DTBook geëxperimenteerd, maar de praktijk viel mij tegen – zo hoopte ik op een ingebouwde rendering van boekachtige element als sidebars, maar dat effect was afwezig totdat je zelf met CSS aan de gang ging. Ook kolommen werkten niet out-of-the-box.

Geheel in strijd met mijn calvinistische opvoeding bewandel ik voorlopig maar de brede weg van XHTML. Lekker vertrouwd, en ook makkelijk om je conversie uit te testen in een browser.

Wat dat laatste betreft: testen in de browser lijkt aardig, maar kan tevens een valkuil blijken te zijn. Tijdens het werken kwam ik er op een gegeven moment achter dat de *position*-properties van CSS niet ondersteund worden in ePub. En daar ging mijn oplossing voor margeteksten. Test daarom, bij het ontwikkelen, regelmatig de stand van zaken in een programma als Adobe Digital Editions en, beter nog, op een aantal echte e-readers.

In de `<!ELEMENT-DTD` is `<article>` de container voor artikelen. In ons voorbeeld maken we van elk artikel een los XHTML-bestand om later in de ePub-file op te nemen. Ook in de praktijk zul je bij je conversie meerdere XHTML-bestanden maken, niet alleen vanwege de beheersbaarheid, maar ook omdat sommige e-readers een groottebeperking hebben – 300KB voor een XHTML-file (tekst + tags, maar de afbeeldingen waarnaar verwezen wordt, tellen niet mee). 300KB is weinig – je bent er snel, zeker als je document tabellen bevat.

Omdat een aflevering van ons mooie tijdschrift tot nu toe gelukkig nog steeds meerdere artikelen bevat, kunnen we in XSLT de constructie zoals getoond in afbeelding 1 toepassen om losse XHTML-files te maken. De namen van die bestanden zijn belangrijk – we moeten later in het TOC-bestand dezelfde namen gebruiken. Het XSLT-fragment in deze afbeelding laat zien hoe je losse bestanden aanmaakt. De functie die de naam berekent, plaats je in de praktijk in een losse file die je in diverse andere stylesheets importeert om de naamgeving van bestanden te centraliseren.

Figure 1.1. Afbeelding 1: het maken van losse XHTML-bestanden

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  version="2.0"
  xmlns:pcm="http://www.masereeuw.nl/2010/xslt/functions"
  exclude-result-prefixes="xs pcm xsl">

  <xsl:param name="OEBPSDir" select="'OEBPS'"/>
  <xsl:param name="TEXTDIR" select="'tekst'"/>

  <xsl:function name="pcm:calculate-html-filename" as="xs:string">
    <xsl:param name="filenum" as="xs:integer"/>
    <xsl:variable name="formattedNum"><xsl:number format="00000"/></xsl:variable>
    <xsl:value-of select="concat($OEBPSDir, '/', $TEXTDIR, '/art'
  </xsl:function>

  <xsl:template match="article">
    <xsl:variable name="filenum"><xsl:number/></xsl:variable>

    <xsl:result-document href="{pcm:calculate-html-filename($file
      encoding="UTF-8" doctype-public="-//W3C//DTD XHTML 1.0 S
      doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-st
    <html>
      <head>
        <title><xsl:value-of select="title"/></title>
        <link rel="stylesheet" type="text/css" href="../
      </head>

      <body><xsl:apply-templates/></body>
    </html>
  </xsl:result-document>
</xsl:template>

  <!-- Schrijf hier zelf de andere templates -->
</xsl:stylesheet>
```

Merk op dat we de XHTML-bestanden plaatsen in de map *OEBPS/tekst* en dat er verwezen wordt naar een CSS-bestand in de map *OEBPS/css*. Deze mappen vormen samen een deel van de mappenstructuur die we uiteindelijk zullen gaan zippen om een ePub-bestand te maken.

De rest van de XML-naar-XHTML-conversie, zoals getriggerd door de aanroep van `<xsl:apply-templates/>` is, zoals dat zo mooi heet, “left as an exercise”.

Noten

Een speciaal probleem vormen voetnoten. Bij de oplossing die mij uiteindelijk het meest bevalt, plaats ik alle voetnoten samen in een apart bestand en genereer ik heen- en weerverwijzingen. Daarbij ga ik er wel vanuit dat de lezer beschikt over een reader die het volgen van hyperlinks ondersteunt.

Een andere oplossing bestaat eruit dat je een noot onderaan een tekstsectie of zelfs de XHTML-pagina plaatst. Dat werkt goed; ik heb deze oplossing uiteindelijk toch verworpen omdat in de gegeven situatie de XHTML-bestanden groter dan 300KB werden, waardoor deze – met de hand, helaas – moesten worden opgeknipt. En bij dat opknippen was het nogal bewerkelijk om de verwijzingen correct te houden.

Nog een andere oplossing bestaat eruit dat je voor elke noot een los bestandje genereert. Die oplossing heeft als nadeel dat bij de e-readers die ik ken, de bladervolgorde niet goed werkt: als de gebruiker een voetnoot leest en op de knop voor de volgende pagina drukt, wordt de volgende voetnoot vertoond, en die kan heel ergens anders in de hoofdtekst worden aangeropen. Eigenlijk zou je willen dat er bij een voetnoot geen “volgende pagina” is na het einde van de noot; de software zou moeten reageren door een boze piep te laten horen of door gewoon niets te doen. De ePub-standaard voorziet ook in iets dergelijks – je kunt in de metadata-file in de verwijzing naar zo'n bestandje (element `<itemref>`) het attribuut `linear="no"` opnemen, maar ik heb geen verschil in verwerking waar kunnen nemen wanneer dit attribuut wel of niet aanwezig was.

Afbeeldingen

Het verwerken van afbeeldingen is niet heel erg moeilijk, maar vereist wat zorgvuldigheid. Om te beginnen moeten alle afbeeldingen worden gekopieerd naar de mappenhiërarchie die we uiteindelijk gaan zippen. Daarbij moet je ervoor zorgen dat je geen afbeeldingen kopieert die niet gebruikt worden (vanwege de omvang van het bestand, en om foutmeldingen bij controle te vermijden). Verder is het niet nodig om hogeresolutie-afbeeldingen te gebruiken.

De scripts die ik zelf gebruik, verzamelen de afbeeldingen uit de gegenereerde XHTML-files. Vervolgens worden ze ontdubbeld en uiteindelijk gekopieerd. Het is uiteraard net zo goed mogelijk om vanuit het bron-XML te werken – de keuze is min of meer arbitrair.

Terzijde: afbeeldingen zijn problematische onderdelen in een e-book, zeker in het geval van echte e-readers met e-paper: kleuren en contrasten vallen weg en er zijn altijd problemen met de maat.

Fonts

In principe kun je je eigen fonts opnemen in je ePub-bestand. Het is de vraag of je dat wilt. Als je tekst nogal wat buitenissige tekens bevat, kan het onvermijdelijk zijn om een font bij te sluiten dat die tekens bevat. In andere gevallen moet je je afvragen wat de meerwaarde is van je eigen fonts: het bestand wordt immers omvangrijker (zeker als je voor vette en cursieve varianten aparte fonts moet insluiten), er spelen copyrightkwesties en de leesbaarheid van het font op e-paper kan een issue zijn.

Feit is wel dat het kan. Indien je besluit om fonts op te nemen, kopieer je deze naar een map in de mappenstructuur; verder maak je ze bekend door ze in de CSS-file te noemen, zoals in het voorbeeld in afbeelding 2:

Figure 1.2. Afbeelding 2: CSS-fragment met font-declaratie

```
@font-face {
  font-family: castricum;
  font-style: normal;
  font-weight: normal;
  src: url(../fonts/castricum-serif.ttf);
}
@font-face {
  font-family: bakkum;
  font-style: normal;
  font-weight: normal;
  src: url(../fonts/bakkum-sans.ttf);
}
body {
  font-family: castricum;
}
```

Tevens zul je de fonts moeten noemen in de metadata-file. Het mimetype van TrueType-fonts is *font/truetype*.

Administratieve bestanden

Naast de inhoudelijke bestanden bevat een ePub-bestand een aantal administratieve bestanden:

- Het metadata-bestand bevat de gebruikelijke metadata als auteur, titel, uitgever en ISBN. Daarnaast bevat het een opsomming van alle meegeleverde bestanden.
- Het TOC-bestand dient om de user-interface te voorzien van een inhoudsopgave. Daarnaast is het mogelijk om een lijst te maken die de lezer een “guided tour” door het document biedt – bijvoorbeeld: alle figuren, alle tabellen, of alle gedichten van Drs. P. Deze laatste functionaliteit (de guided tour) heb ik echter nog niet ondersteund gezien in de e-readers die ik ken.

Het metadatabestand

Het metadatabestand (*metadata.opf*, maar de naam is vrij) is het bestand waarnaar de startfile van het ePub-bestand (*container.xml*) verwijst. Het bestaat uit drie hoofdonderdelen:

- `<metadata>` - de echte metadata;
- `<manifest>` - hier wordt aan elk bestand een mime-type en een id gekoppeld;
- `<spine>` - deze bestaat uit verwijzingen naar de in het manifest gedefinieerde id's. De volgorde in de spine bepaalt de leesvolgorde; als een bestand (zoals een afbeelding of een noot) buiten de normale leesvolgorde staat, ken je het attribuut *linear="no"* toe (dit lijkt echter in de praktijk voor teksten geen effect te hebben, zoals hierboven al is opgemerkt).

Afbeelding 3 geeft een voorbeeld van de hoofdstructuur van het metadata-bestand:

Figure 1.3. Afbeelding 3: de hoofdstructuur van het bestand *metadata.opf*

```
<?xml version="1.0" encoding="UTF-8"?>
▼ <package xmlns="http://www.idpf.org/2007/opf" xmlns:xs="http://www
  version="2.0" unique-identifier="mijn-id">
▼   <metadata xmlns:opf="http://www.idpf.org/2007/opf" xmlns:dc="h
     <dc:title>Maak thuis Uw eigen e-book in Uw vrije tijd zonde
     <dc:creator opf:role="aut">Pieter Masereeuw</dc:creator>
     <dc:language>nl</dc:language>
     <dc:identifier id="mijn-id" opf:scheme="ISBN">0000000000000
     <dc:rights>Copyright © 2010 Pieter Masereeuw</dc:rights>
   </metadata>
▶   <manifest> [33 lines]
▶   <spine toc="ncx"> [29 lines]
</package>
```

In afbeelding 4 zien we hoe je met XSLT de manifest- en spinesecties genereert:

Figure 1.4. Afbeelding 4: de manifest- en spinesecties van *metadata.opf*

```
<xsl:function name="pcm:calculate-html-id" as="xs:string">
  <xsl:param name="filenum" as="xs:integer"/>
  <xsl:value-of select="concat('tekst.', $filenum)"/>
</xsl:function>

<xsl:template match="/element">
  <package xmlns="http://www.idpf.org/2007/opf" version="2.0">
    <metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
      <manifest>
        <item id="ncx" href="toc.ncx" media-type="application/x-dtb+xml"/>
        <item id="style" href="css/element.css" media-type="text/css"/>
        <xsl:for-each select="article">
          <xsl:variable name="filenum"><xsl:number/></xsl:variable>
          <item id="{pcm:calculate-html-id($filenum)}"
            href="{pcm:calculate-html-filename($filenum)}"
            media-type="application/xhtml+xml"/>
        </xsl:for-each>
      </manifest>
      <spine toc="ncx">
        <xsl:for-each select="article">
          <xsl:variable name="filenum"><xsl:number/></xsl:variable>
          <itemref idref="{pcm:calculate-html-id($filenum)}"/>
        </xsl:for-each>
      </spine>
    </package>
  </xsl:template>
```

Uit afbeelding 4 valt niet af te lezen hoe je de afbeeldingen opneemt. Daar zijn verschillende mogelijkheden voor. Ik heb ervoor gekozen om de afbeeldingen buiten XSLT te ontdebelen; de ontdebeldde afbeeldingsnamen worden in een tijdelijk XML-bestandje geplaatst. Dat XML-bestandje wordt ingelezen met behulp van de *document()*-functie van XPath. In principe kun je het ontdebelen ook in XSLT doen, maar omdat ik in mijn scripts ook nog bestanden moest kopiëren, was de genoemde oplossing het makkelijkst.

Lastig bij afbeeldingen is dat je het mime-type moet meegeven. Om die te bepalen gebruik ik de bestandsextensie, een voor de hand liggende oplossing, die wat geknutsel met reguliere expressies vereist. Zie afbeelding 5. Verondersteld is dat de code uit afbeelding 5 wordt ingevoegd na de eerste *<xsl:for-each>* van afbeelding 4. Opname van de afbeeldingen in de spine is niet nodig als er vanuit de XHTML-files naar wordt verwezen.

Figure 1.5. Afbeelding 5: afbeeldingen opnemen in *metadata.opf*

```
<manifest>
  <!-- ... -->
  <xsl:for-each select="article"> [5 lines]
    <xsl:for-each select="document('/tmp/afbeeldingen.
      <xsl:variable name="num"><xsl:number format="0
      <xsl:variable name="extension" as="xs:string">
        <xsl:analyze-string select="." regex=".*[.
          <xsl:matching-substring>
            <xsl:value-of select="lower-case(r
          </xsl:matching-substring>
        </xsl:analyze-string>
      </xsl:variable>
    <xsl:variable name="mediatype" as="xs:string">
      <xsl:choose>
        <xsl:when test="$extension = 'gif'">im
        <xsl:when test="$extension = 'jpg'">im
        <xsl:when test="$extension = 'jpeg'">i
        <xsl:when test="$extension = 'png'">im
        <xsl:when test="$extension = 'tif'">im
        <xsl:when test="$extension = 'tiff'">i
        <xsl:otherwise>image/unknown</xsl:othe
      </xsl:choose>
    </xsl:variable>
    <item id="{concat('afb.', $num)}" href="{conca
      media-type="{${mediatype}"/>
    </xsl:for-each>
</manifest>
```

Het TOC-bestand

Het TOC-bestand (*toc.ncx*, maar de naam is vrij) bestaat uit drie verplichte delen: de *<head>*, de *<docTitle>* en de *<navMap>*. Op de *<navMap>* kun je nog een aantal *<navList>*-elementen laten volgen. Die elementen vormen de reeds eerder genoemde “guided tours”, maar omdat die noch in de belangrijkste e-readers, noch in Adobe Digital Editions ondersteund worden, gaan we daar niet verder op in.

Afbeelding 6 toont de eerste twee delen van het TOC-bestand.

Figure 1.6. Afbeelding 6: de eerste twee secties van bestand toc.ncx

```
<?xml version="1.0" encoding="UTF-8"?>
<ncx xmlns="http://www.daisy.org/z3986/2005/ncx/" version="2005-11-18">
  <head>
    <meta name="dtb:uid" content="00000000000000" />
    <meta name="dtb:depth" content="'3'" />
    <meta name="dtb:generator" content="Demo-scripts van Pieter" />
    <meta name="dtb:totalPageCount" content="0" />
    <meta name="dtb:maxPageNumber" content="0" />
  </head>
  <docTitle>
    <text>Maak thuis Uw eigen ebook in Uw vrije tijd zonder schroeven</text>
  </docTitle>
  <navMap> [86 lines]
</ncx>
```

Het interessantste deel van het TOC-bestand is natuurlijk de `<navMap>`-sectie. Zoals uit afbeelding 7 blijkt, is dit een geneste structuur die gevormd wordt door `<navPoint>`-elementen. Elk navigatiepunt bevat een label met tekst die in de inhoudsopgave van het programma van de e-reader vertoond wordt (zorg voor een goede behandeling van whitespace!).

Ieder `<navPoint>`-element heeft een uniek id. Dit id is een hulpmiddel voor software die verwijzingen naar navigatiepunten wil vasthouden, zoals bookmarks. Verder heeft elk `<navPoint>`-element een verplicht `playOrder`-attribuut. Dit attribuut bevat het volgnummer van elk navigatiepunt. Er mogen geen nummers ontbreken en de nummers moeten in de juiste volgorde staan. Het `playOrder`-attribuut is volgens mij op dit moment weliswaar verplicht, maar nog niet nuttig. Het is van belang op het moment dat je van de functionaliteiten van `<navLists>`'s gebruik kunt maken: op dat moment moet de e-readersoftware zonder al te veel ingewikkelde processing kunnen beslissen op welke plaats de lezer zich in het document bevindt, als hij of zij besluit om de guided tour te verlaten.

De TOC van afbeelding 7 wordt in XSLT samengesteld door alle artikelen en (geneste) secties te selecteren. De secties worden voorzien van een id (in de tekst) en een id-ref (in de TOC). De waarde van het `playorder` attribuut komt tot stand door middel van `<xsl:number count="article | section" level="any"/>` (`level="any"` is hier cruciaal).

De echte verwijzing naar de content vindt plaats met het `<content src="..." />`, waarbij de waarde van `src` een hyperlink is die eventueel van een anker (hekje gevolgd door id) is voorzien.

Afbeelding 7 zal een boel duidelijk maken.

Figure 1.7. Afbeelding 7: geneste navigatiepunten in bestand *toc.ncx*

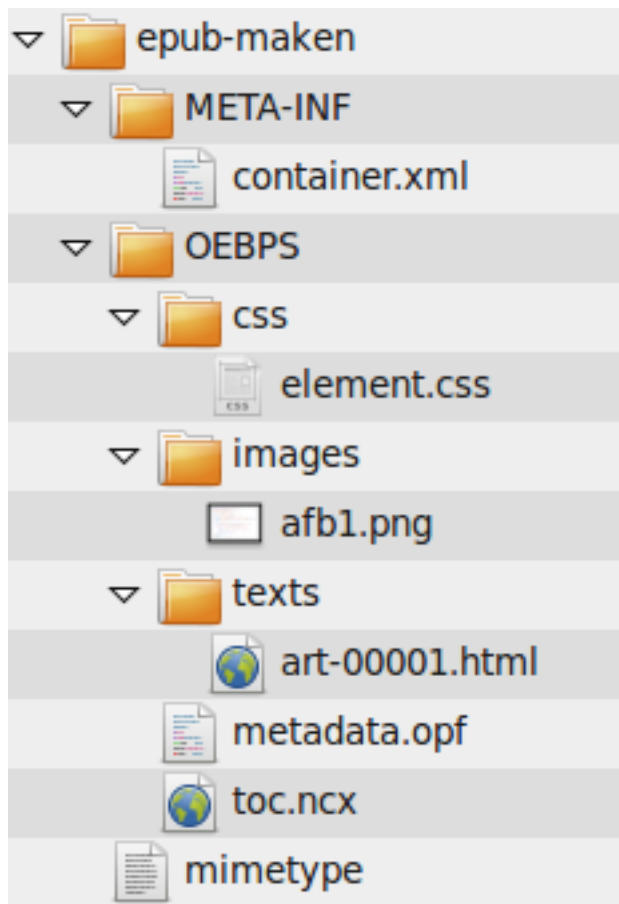
```
<?xml version="1.0" encoding="UTF-8"?>
<ncx xmlns="http://www.daisy.org/z3986/2005/ncx/" version="2005-11-18">
  <head> [6 lines]
  <docTitle> [2 lines]
  <navMap>
    <navInfo><text>Hoofdstukken</text></navInfo>
    <navPoint id="navpoint.1" playOrder="1">
      <navLabel>
        <text>Maak thuis Uw eigen ebook in Uw vrije tijd zonder schroeven</text>
      </navLabel>
      <content src="texts/hs-00001.html"/>
    <navPoint id="navpoint.2" playOrder="2">
      <navLabel>
        <text>Voor we beginnen</text>
      </navLabel>
      <content src="texts/hs-00001.html#section-1"/>
    </navPoint>
    <navPoint id="navpoint.3" playOrder="3">
      <navLabel>
        <text>Wat zit er in een ePub-bestand</text>
      </navLabel>
      <content src="texts/hs-00001.html#section-2"/>
    </navPoint>
    <navPoint id="navpoint.4" playOrder="4"> [29 lines]
    <navPoint id="navpoint.9" playOrder="9"> [23 lines]
    <navPoint id="navpoint.13" playOrder="13"> [5 lines]
    <navPoint id="navpoint.14" playOrder="14"> [5 lines]
  </navMap>
</ncx>
```

Voor de hier verder niet behandelde *<navList>*-elementen geldt overigens dat ze geen nesting kennen, en dat de *playOrder*-attributen niet aaneensluitend hoeven te zijn.

De resterende lijm: *mimetype* en *META-INF/container.xml*.

Onze mappenstructuur ziet er nu uit als in afbeelding 8:

Figure 1.8. Afbeelding 8: de mappenstructuur van een ePub-bestand



De file *mimetype* hebben we hierboven al genoemd – deze bevat alleen maar de tekst *application/epub+zip*. De file *META-INF/container.xml* is niet spannend, maar wel belangrijk. Dit bestand vertelt de e-reader waar de OPF-file staat. Zoals u in afbeelding 9 kunt zien, bevat deze file een element `<rootfiles>` en een element `<rootfile>`. Dit suggereert dat er meerdere rootfiles kunnen zijn en dat is ook zo: *container.xml* kan naar naar meerdere plekken verwijzen – naast een OPF-file zou je bijvoorbeeld ook een PDF-file kunnen noemen, om als alternatief medium te dienen. Ik heb dit overigens nooit in de praktijk getest, dus ik weet niet in hoeverre de verschillende leesapplicaties dit ondersteunen.

Figure 1.9. Afbeelding 9: de file *container.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<container version="1.0" xmlns="urn:oasis:names:tc:opendocument:xml1.0">
  <rootfiles>
    <rootfile full-path="OEBPS/metadata.opf" media-type="application/epub+zip">
    </rootfile>
  </rootfiles>
</container>
```

Het aanmaken van het ZIP-bestand

Het zip-bestand dient niet de gebruikelijke extensie *.zip* te hebben, maar *.epub*. Je maakt het bestand niet met een standaard compressietool, want de volgorde is belangrijk: de file *mimetype* moet aan het begin staan en mag niet worden gecomprimeerd.

Met behulp van het zip-commando dat met veel Linux-distributies wordt meegeleverd (en dat vermoedelijk ook wel elders beschikbaar is), kun je op de volgende manier een ePub-bestand aanmaken (verondersteld is dat de huidige map de root is van de ePub-map):

```
EPUBFILE=../element.epub
```

```
zip -q "$EPUBFILE" -X -D -0 mimetype
```

```
zip -q -g "$EPUBFILE" -X -D -r -9 META-INF OEBPS
```

Voor meer details verwijst ik naar de man-page.

De epub-checker

Om na te gaan of het ePub-bestand aan de normen voldoet, kun je gebruik maken van de ePub-checker die je vinden bij Google Code. Dit tool wordt in de verwijzingen hieronder genoemd. Helaas doet het tool geen controle van de CSS-file.

Verwijzingen

- <http://www.idpf.org/specs.htm>
- <http://www.openebook.org/oebps/oebps1.2/download/oeb12-xhtml.htm>
- http://www.idpf.org/2007/opf/OPF_2.0_final_spec.html
- <http://www.niso.org/workrooms/daisy/Z39-86-2005.html#NCX>
- <http://code.google.com/p/epubcheck/>

Pieter Masereeuw is zelfstandig uitgeeftechnoloog, als dat tenminste een woord is. En anders is hij het ook. Hij houdt zich graag bezig met Open Source-oplossingen, XML, Java en XSLT.