Optimize Your Digital Books

EPUB 3 Best Practices

Matt Garrish &
Markus Gylling

O'REILLY® | <idpf>

Digital Publishing

EPUB 3 Best Practices

Ready to take your ebooks to the next level with EPUB 3? This concise guide includes best practices and advice to help you navigate the format's wide range of technologies and functionality. EPUB 3 is set to turn electronic publishing on its head with rich multimedia reading experiences and scripted interactivity, but this specification can be daunting to learn. This book provides you with a solid foundation.

Written by people involved in the development of this specification, *EPUB 3 Best Practices* includes chapters that cover unique aspects of the EPUB publishing process, such as technology, content creation, and distribution.

- Get a comprehensive survey of accessible production features
- Learn new global language-support features, including right-to-left page progressions
- Embed content with EPUB 3's new multimedia elements
- Make your content dynamic through scripting and interactive elements
- Work with publication and distribution metadata
- Create synchronized text and audio playback in reading systems
- Learn techniques for fixed and adaptive layouts

Matt Garrish, owner of CData and editor of the EPUB 3 revision, has worked closely with CNIB and the DAISY Consortium to help make books more accessible. He is the author of *What is EPUB 3?* (O'Reilly).

Markus Gylling, CTO of the DAISY Consortium and the International Digital Publishing Forum, serves as chair of the EPUB 3 Working Group. He led the development of the EPUB 3 specification in 2011.

"EPUB 3 Best Practices *is the essential resource covering all the key capabilities of EPUB 3, the next-generation portable document format built on HTML5 and Web Standards. Whether you're an accomplished expert or a newcomer to digital publishing, this book will help you create engaging, accessible digital publications that go way beyond plain text and print-replica editions."*

—Bill McCoy
Executive Director, International Digital Publishing Forum (IDPF)

US $29.99 CAN $31.99
ISBN: 978-1-449-32914-3
52999
9 781449 329143

CHAPTER 1

Package Document and Metadata

Bill Kasdorf

Vice President, Apex CoVantage

One of the most common misconceptions about EPUB is that it is a “flavor” of XML. (“Should I use EPUB or DocBook?” or, even worse, “Should I use EPUB or HTML5?” *Hint*: EPUB (pretty much) = HTML5.) Due partly to the convenient single-file format provided as *.epub*, people sometimes fail to realize that EPUB is not just, and not mainly, a specification for the markup of content documents. It is a *publication format*, and as such it specifies and documents a host of things that publications need to include—content documents, style sheets, images, media, scripts, fonts, and more, as discussed in detail in the other chapters of this book. In fact, EPUB is sometimes thought of as “a website in a box,” though it is actually much more than that.

What is arguably the most important thing about it is this: it *organizes* all the stuff in the box. It’s designed to enable reading systems to easily and reliably know, up front, what’s contained in a given publication, where to find each thing, what to do with it, how the parts relate to each other. And it enables publishers to provide that information in one clear, consistent form that all reading systems should understand, rather than in different, proprietary ways for each recipient system.

This, of course, is what metadata is for: it’s not the content, it’s information about the content. EPUB 3 accommodates much richer metadata than EPUB 2 did, and it enables that metadata to be associated not just with the publication as a whole, but also with individual components of the publication and even with elements within the content documents themselves. While it doesn’t *require* much more than EPUB 2 did (in the interest of backward compatibility), it *accommodates* the much richer metadata that makes publications so much more discoverable and dynamic, so much more usable and useful.

The place where all this information is organized is the *package document*, an XML file that is one of the fundamental components of an EPUB, the *.opf* file. (The extension *.opf* stands for Open Package Format, which was the precursor to the new Publications specification.) In addition to containing most of the EPUB's metadata, the package document serves as a hub that associates that metadata with the other resources comprising the EPUB. All of this is then literally "zipped up" in a single-file container, the *.epub* file. *Voilá*, the "website in a box"—but one with a complete packing list and indispensable assembly instructions that ensure that an EPUB 3–compliant reading system will deliver the publication properly to the end user.

Before we take the lid off the box, let's look at the basic building blocks of EPUB 3 metadata.

Vocabularies

In order to make EPUBs easy to create, very little metadata is actually required, and the requirements are almost identical to those in EPUB 2. Like EPUB 2, EPUB 3 uses the Dublin Core Metadata Element Set (DCMES) for much of its required and optional metadata. Commonly referred to as "Dublin Core," DCMES is widely used as a basic framework for metadata of all sorts, from publication metadata to metadata for media like movies, audio, and images. You'll see examples throughout the balance of this chapter.

But EPUBs need to handle richer metadata as well, both to provide important information to the reading system and the end user, and to enable the more sophisticated functionality EPUB 3 offers. This simplicity-plus-complexity dilemma is addressed by providing:

- A basic *default vocabulary* that all EPUB 3 reading systems are required to understand;
- A short list of *reserved vocabularies* that can be used with their standard prefixes without declaration; and
- A mechanism by which any other vocabulary and its prefix can be declared, along with a pointer to where the authoritative definition of that vocabulary (in either human-readable or machine-readable form) can be found.

It's important to realize that this is not just designed to make it easy to create EPUBs; equally important is that it is designed to make it easy for reading systems to process EPUBs. While EPUB 3 enables full-blown metadata records like ONIX files for distribution and MARC records for cataloguing to be provided, such records are rich, complex, and can be used quite differently by different publishers.

ONIX is a good example of that: it provides for literally hundreds of different features and codes by which book supply chain metadata can be described; no publisher uses all of it, and different publishers make different choices as to what to use. This is very useful to the publisher who needs to convey that metadata to a recipient who knows how to handle it, but it is too much to ask all EPUB 3 reading systems to be able to handle. Plus, that standard changes frequently as more terms and features are added. And EPUB 3 is not just for books; many publishers who create EPUBs don't use ONIX at all.

EPUB 3 metadata, by contrast, is designed to provide a clear, consistent foundation, describing metadata that all EPUB 3 reading systems can be expected to handle, for all types of content, and clearly specifying which things are optional. So while you can include an ONIX file or MARC record if you want to, for the EPUB 3 metadata itself, you need to follow EPUB 3's rules. That's what this chapter is all about.

The Default Vocabulary

The basic vocabulary on which EPUB 3 metadata depends is simple but powerful. It provides specific, clearly defined terms that are used to describe fundamental properties of key elements:

`meta`
: The workhorse of EPUB 3 metadata

`link`
: Enabling the inclusion of external resources

`item`
: Providing metadata about each item in the `manifest`

`itemref`
: For metadata associated with items in the `spine`

These default vocabulary terms are specific to each of those elements and provide reading systems with a reliable, consistent way to understand how to handle each of them. For example, some of the default vocabulary terms for `item` in the `manifest` provide a means to alert the reading system to which files include MathML or SVG, and to identify which file is the cover image. One of the default vocabulary terms for `link` identifies that the resource being linked to is an ONIX record. The default vocabulary terms for each of these components are discussed in more detail below.

The Reserved Vocabularies

The reserved vocabularies provide commonly used sets of terms that can be used, with the proper prefix, without requiring those prefixes to be declared in the EPUB. In other words, the reading system is supposed to know where to find authoritative documentation of these vocabularies.

The four vocabularies reserved in EPUB 3.0 are:

`dcterms`
: A richer but more restrictive counterpart to DCMES in Dublin Core, designed to enable "linked data"

`marc`
: A vocabulary commonly used by libraries for bibliographic metadata

`media`
: The vocabulary on which EPUB 3's Media Overlays specification depends

`onix`
: The vocabulary used for book supply chain metadata

The prefix *xsd* is also reserved for defining W3C XML Schema data types.

Using Other Vocabularies

Of course, there are many more vocabularies that are useful to publishers, and new ones are being created all the time. Ideally, these are public standards for which authoritative documentation can be referenced. But in order to be as flexible as possible, EPUB 3 even permits proprietary vocabularies to be used.

To use any of these other vocabularies, their terms must include a *prefix* (similar to how namespaces work), and each such prefix used in an EPUB must be declared in the `prefix` attribute of the `package` element, which is the root container of the package document (more on that below). This is done by "mapping" each prefix to a URI (Uniform Resource Identifier) that tells where its vocabulary is documented. Examples commonly used by publishers include:

`xmp`
: The Extensible Metadata Platform, widely used for metadata about images and other media:

```
prefix="xmp: http://ns.adobe.com/xap/1.0/"
```

`prism`
: The very rich vocabulary used for magazine and other publication metadata:

```
prefix="prism: http://prismstandard.org/namespaces/basic/3.0/"
```

custom
: A proprietary metadata scheme used by a publisher:

```
prefix="TimeInc: http://www.timeinc.com/PRISM/2.1/"
```

And what about EPUB 3's default vocabulary? That is both the simplest and, potentially, the most complicated of all.

The All-Powerful meta Element

The workhorse of EPUB 3 metadata is the `meta` element, which provides a simple, generic, and yet surprisingly flexible and powerful mechanism for associating metadata of virtually unlimited richness with the EPUB package and its contents. An EPUB can have any number of `meta` elements. They're contained in the `metadata` element, the first child of the `package` element, and from that central location they serve as a hub for metadata about the EPUB, its resources, its content documents, and even locations within the content documents.

Here's how it works.

The `meta` element uses the `refines` attribute to specify what it applies to, using an ID in the form of a relative IRI. So, for example, a `meta` element can tell you something about chapter 5:

```
<meta refines="#[ID of chapter 5]">...</meta>
```

or about the author's name:

```
<meta refines="#creator">...</meta>
```

or about a video:

```
<meta refines="#video3">...</meta>
```

When the `refines` attribute is not provided, it is assumed that the `meta` element applies to the package as a whole; this is referred to as a *primary expression*. When the `meta` element does have a `refines` attribute, it is called a *subexpression*.

Each `meta` has a `property` attribute that defines what kind of statement is being made in the text of the `meta` element. The values of `property` can be the default vocabulary, a term from one of the reserved vocabularies, or a term from one of the vocabularies defined via the prefix mechanism. For example, you can provide the author Haruki Murakami's name in Japanese like this:

```
<meta refines="#creator"
property="alternate-script"
xml:lang="ja">
  村上 春樹
</meta>
```

The default vocabulary for `meta` consists of the following property values:

`alternate-script`
: Typically used to provide versions of titles and the names of authors or contributors in a language and script identified by the `xml:lang` attribute, as shown in the previous example.

`display-seq`
: Used to specify the sequence in which multiple versions of the same thing—for example, multiple forms of the title—should be displayed:

```
<meta refines="#title2" property="display-seq">1</meta>
```

`file-as`
: Provides an alternate version—again, typically of a title or the name of an author or other contributor—in a form that will alphabetize properly, e.g., last-name-first for an author's name or putting "The" at the end of a title that begins with it:

```
<meta refines="#creator" property="file-as">Murakami, Haruki</meta>
```

`group-position`
: Specifies the position of the referenced item in relation to others that it is grouped with. This is useful, for example, so that all the titles in a series are displayed in proper order in a reader's bookshelf:

```
<meta refines="#title3" property="group-position">2</meta>
```

`identifier-type`
: Provides a way to distinguish between different types of identifiers (e.g., ISBN versus DOI). Its values can be drawn from an authority like the ONIX Code List 5, which is specified with the `scheme` attribute:

```
<meta refines="#src-id"
property="identifier-type"
scheme="onix:codelist5">
  15
</meta>
```

`meta-auth`
: Documents the "metadata authority" responsible for a given instance of metadata:

```
<meta refines="isbn-id" property="meta-auth">isbn-international.org</meta>
```

`role`
: Most often used to specify the exact role performed by a contributor—for example, a translator or illustrator:

```
<meta refines="#creator" property="role" scheme="marc:relators">ill</meta>
```

`title-type`
: Distinguishes six specific forms of titles (see "Types of Titles" (page 14)):

```
<meta refines="#title" property="title-type">subtitle</meta>
```

A `meta` element may also have an ID of its own, as the value of the `id` attribute:

```
<meta refines="isbn-id"
property="meta-auth"
id="meta-auth">
  isbn-international.org
</meta>
```

This ID can be used to make metadata chains, where one `meta` refines another. The element may also have a formal identifier of the scheme used for the value of the property (using the `scheme` attribute).

You can also use property values, which must include the proper prefix, from any of the reserved vocabularies or any vocabulary for which you've declared the prefix:

```
<meta property="dcterms:dateCopyrighted">2012</meta>
```

You'll notice that the previous example did not include a `refines` attribute. This was intentional, as the other use for the `meta` element is to define metadata for the publication as a whole. We'll look at the Dublin Core elements for publication metadata shortly, but you are not limited to using them. If another vocabulary provides richer metadata, you can use the `meta` element to express it.

You will see examples of the `meta` element throughout this chapter. While it is a bit abstract and thus can be hard to grasp at first, once you get the hang of it you'll find it to be easy to use, and indispensable, for enriching and empowering your EPUB with metadata.

Publication Metadata

Most of the metadata in a typical EPUB is associated with the publication as a whole. (An exception is an EPUB of an issue of a magazine, where most of the metadata is at the article, or content document, level; see "Types of Titles" (page 14).) This is intended to tell a reading system, when it opens up the EPUB, everything it needs to know about what's inside. Which EPUB is this (*identifiers*)? What names is it known by (*titles*)? Does it use any vocabularies I don't necessarily understand (*prefixes*)? What language does it use? What are all the things in the box (*manifest*)? Which one is the cover image, and do any of them contain MathML or SVG or scripting (*spine itemref properties*)? In what order should I present the content (*spine*), and how can a user navigate this EPUB (*the nav document*)? Are there resources I need to link to (*link*)? Are there any media objects I'm not designed by default to handle (*bindings*)?

Having all of this information up-front in the EPUB makes things much easier for a reading system, rather than requiring it to simply discover that unrecognized vocabulary, or that MathML buried deep in a content document, only when it comes across it, as a browser does with a normal website.

We'll take a look at each of these, followed by a deeper dive into some of the more interesting ones.

The Package Document Structure

An EPUB provides almost all of this fundamental information in an XML file called the package document. This contains that invaluable packing list and those indispensable assembly instructions that enable a reading system to know what it has and what to do with it.

The root element of the package document is the `package` element. This, in turn, contains the metadata and resource information in its child elements, in this order:

- `metadata` (required)
- `manifest` (required)
- `spine` (required)
- `guide` (optional and deprecated; a carryover from EPUB 2)
- `bindings` (optional)

The following markup shows the typical structure you'll find:

```
<package ... version="3.0" xmlns="http://www.idpf.org/2007/opf">
   <metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
      ...
   </metadata>
   <manifest>
      ...
   </manifest>
   <spine>
      ...
   </spine>
</package>
```

In addition to declaring the namespace on the root `package` element, you must also declare the `version` (EPUB 3s must declare `3.0`). The following attributes are also recommended:

`xml:lang`
: The language of the package document (not necessarily the same as the publication!)

`dir`
: The text directionality of the package document: left-to-right (`ltr`) or right-to-left (`rtl`)

The metadata Element

The `metadata` element contains the same three required elements as it did in EPUB 2, one new required element, and a number of optional elements, including that all-powerful `meta` element described previously.

As mentioned earlier, EPUB continues to use the Dublin Core Metadata Element Set (DCMES) for most of its required and optional metadata.

XML rules require that you declare the Dublin Core namespace in order to use the elements. This declaration is typically added to the `metadata` element, but can also be added to the root `package` element. For example:

```
<metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
```

The required elements, which can occur in any order, are the following:

`dc:identifier`
: Contains an identifier for the publication. An EPUB can have any number of these (for example, an ISBN, a DOI, and even a proprietary identifier), but it must have at least one. And one must be designated as the *unique identifier* for the publication by the `unique-identifier` attribute on the root `package` element. (In a departure from EPUB 2, this is not, however, a unique *package* identifier; see "Identifiers" (page 11) for more on this.) A `dc:identifier` in `metadata` may or may not have an `id` attribute; the `id` is only required for the one designated as the publication's unique identifier.

`dc:title`
: Contains a title for the publication. Like `dc:identifier`, there can be more than one of these, but there must be at least one. While an `id` is not required on `dc:title`, it is a good idea to provide one, in order to associate metadata with it; you'll see why this is useful in "Types of Titles" (page 14). In addition, the optional `xml:lang` attribute enables the language of a title to be specified, and the optional `dir` attribute specifies its reading direction, with values of `ltr` (left-to-right) and `rtl` (right-to-left).

`dc:language`
: Specifies the language of the publication's content. (You can specify the language of many *metadata* elements with the `xml:lang` attribute; this `dc:language` element on `metadata` is about the *content* of the EPUB.) There can be more than one—for example, an EPUB might mainly be in English but have sections in French—but there must be at least one. And languages must be specified with the scheme provided in RFC5646, "Tags for Identifying Languages"; you can't just say "French."

Here is how you would express the required metadata for this book:

```
<dc:identifier id="pub-identifier">urn:isbn:9781449325299</dc:identifier>
<dc:title id="pub-title">EPUB 3 Best Practices</dc:title>
<dc:language id="pub-language">en</dc:language>
```

All of the other elements in the Dublin Core Metadata Elements Set (DCMES) are optional, but many of them are quite useful. These are `dc:contributor`, `dc:coverage`, `dc:creator`, `dc:date`, `dc:description`, `dc:format`, `dc:publisher`, `dc:relation`, `dc:rights`, `dc:source`, `dc:subject`, and `dc:type`. You'll see these in many of the examples in this chapter. They may all have optional `id`, `xml:lang`, and `dir` attributes. The ones most publishers will be likely to use are these:

`dc:creator`
: Contains the name of a person or organization with primary responsibility for creating the content, such as an author; `dc:contributor` is used in the same way, but indicates a secondary level of involvement (for example, a translator or an illustrator). The EPUB default vocabulary for properties can be used to provide further information, using that workhorse `meta` mechanism described above. For example, `property="role"` can be used to specify that a contributor was the translator, and `property="file-as"` can be used to provide her name in last-name-first form so it will sort properly alphabetically:

```
<dc:creator id="author">Bill Kasdorf</dc:creator>
<meta refines="#author" property="role" scheme="marc:relators">aut</meta>
```

`dc:date`
: Used to provide *the date of the EPUB publication*, not the publication date of a source publication, such as the print book from which the EPUB has been derived. Only one `dc:date` is allowed. Its content should be provided in the standard W3C date and time format, for example:

```
<dc:date>2000-01-01T00:00:00Z</dc:date>
```

`dc:source`
: Contains the identifier of the source publication from which the EPUB was derived, such as the print version. Only one `dc:source` is allowed:

```
<dc:source id="src-id">urn:isbn:9780375704024</dc:source>
```

`dc:type`
: Presents a bit of a curveball at the moment, because the IDPF has not yet defined values for it. It is intended to distinguish specialized types of EPUBs like dictionaries or indexes. Since it's optional, it may be best not to use it until there is a standard set of values available.

The `metadata` element can also contain any number of those useful `meta` elements described in "The All-Powerful meta Element" (page 5). An EPUB with rich metadata is likely to include lots of them, each one with its `refines` attribute identifying what its `property` attribute applies to, its optional `id` enabling *itself* to subsequently be refined by another `meta`, and its optional `scheme` documenting a formal definition of the property it describes. This is deliberately generic and abstract: in order to enable you to use virtually any kind of metadata in an EPUB, it specifies nothing but this bare-bones mechanism. Users often look in vain for more specifics at first; it is only after you begin to use `meta` that you come to realize its flexibility and power.

There is one very specific use of the `meta` element that is quite important; in fact, it is a requirement for EPUB 3. The `meta` element is used to provide a *timestamp* that records the *modification date* on which the EPUB was created. It uses the `dcterms:modified` property and requires a value conforming to the W3C dateTime form, like this:

```
<meta property="dcterms:modified">2011-01-01T12:00:00Z</meta>
```

When used with the *unique identifier* that identifies the publication, this further identifies the package.

More on this later in "Identifiers" (page 11).

Mention should be made here of the `meta` element as defined in the previous EPUB specification, OPF2. That OPF2 version of `meta` has been replaced by the new definition in EPUB 3. However, despite the fact that it is obsolete, it is still permitted in an EPUB so that EPUB 3 reading systems don't reject older EPUB 2s—but they're required to ignore those obsolete OPF2-style `metas`. We'll see a use for this element when we look at covers in "Covers" (page 85) in Chapter 3.

Finally, the `metadata` element can also include `link` elements. These are designed to associate resources with the publication that are not a part of its direct rendering. Unlike most publication resources, linked resources can be provided either within the container or outside it. The element is primarily designed to enable metadata records of different types to be included in an EPUB.

The `link` element, along with the `bindings` element that is a sibling, rather than a child, of `metadata`, is discussed in more detail later in "Links and Bindings" (page 20).

Identifiers

Andy Tanenbaum's joke about standards, "The nice thing about standards is that there are so many of them to choose from," applies just as well to identifiers. The ironic implication of the joke (shouldn't one standard, one identifier, be sufficient?) turns out to be far from the truth. Identifiers have different purposes: the ISBN is a product identifier, the ISTC identifies textual works, the DOI provides an "actionable" and persistent identifier, the ISSN identifies a serial publication; and publishers typically have proprietary identifiers for their publications as well. Many of these can apply to a given EPUB.

Providing these identifiers—and, ideally, documenting them properly—uses a combination of the Dublin Core `dc:identifier` and EPUB 3's `meta` element. Here's an example from the EPUB 3 specs:

```
<metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
    <dc:identifier id="pub-id">
      urn:doi:10.1016/j.iheduc.2008.03.001
    </dc:identifier>
    <meta refines="#pub-id"
    property="identifier-type"
    scheme="onix:codelist5">
      06
    </meta>
    ...
</metadata>
```

Since we can have any number of these, we need to give this one an ID; we've named it `pub-id`. Then the content of `dc:identifier` is the identifier itself, which in this case is a *digital object identifier* (DOI).

While it might seem obvious that that identifier is a DOI (it does begin with `doi:`, after all), that is not true of every possible identifier we might want to use. In the interest of making things as clear and explicit as possible (for either human or machine interpretation), we need to identify what kind of identifier that is and where its authoritative definition can be found. That's what the `meta` element is doing. It says, "I'm refining the element I designated as `pub-id`; what I'm telling you about it is what type of identifier it is; and the type of identifier is the one described as item 06 in ONIX Codelist 5." While a reading system is not, of course, required to go and consult ONIX Codelist 5, there is a clear, unambiguous record in the EPUB metadata of exactly what kind of identifier this one is. ONIX Codelist 5 provides a convenient, authoritative reference to types of identifiers; but if this were a publisher's own proprietary identifier (a common type of identifier a publisher might want to include), then it could simply say `scheme="proprietary"`.

As mentioned previously, an EPUB 3 can have any number of `dc:identifier` elements in its `metadata`. And one of them must be designated, via the `unique-identifier` attribute on the root `package` element, as the unique identifier of the publication. Isn't this the same as saying it's the unique identifier of the EPUB, just as EPUB 2 specified?

It turns out that the meaning of *unique* is not "unique." When technologists—or reading systems—say an identifier uniquely identifies an EPUB, they mean it quite literally: if one EPUB is not bit-for-bit identical to another EPUB, it needs a different unique identifier, because it's not the same thing; systems need to tell them apart. Publishers, on the other hand, want the identifier to be persistent. To them, a new EPUB that corrects some typographical errors or adds some metadata is still "the same EPUB"; giving it a different identifier creates ambiguity and potentially makes it difficult for a user to realize that the corrected EPUB and the uncorrected EPUB are really "the same book."

After quite a bit of struggle, the EPUB 3 Working Group came up with an elegant solution to this dilemma by doing two simple things: changing the definition of *unique identifier* and adding the *timestamp* mentioned earlier.

The specifications for EPUB 3 say that the *unique identifier*—the value of the `unique-identifier` attribute on the `package`—should be persistent in terms of the *publication*. It's a *publication identifier*. It should not change when the only differences between the old and new versions of the EPUB are minor changes like additions to metadata or fixing errata. (New editions, on the other hand, or derivative versions of various sorts, like a translation or even an illustrated version of a previously nonillustrated text, obviously must get a new "unique identifier.")

But the EPUB 3 specs also require the `package` to contain a `meta` element that records the date and time, via the timestamp, when that EPUB file was created. It is the combination of these two things, the publication identifier and the timestamp, that serves as the package identifier that tells the reading system exactly which EPUB file it is dealing with.

So although the exact meaning of "unique" is still fuzzy—two EPUBs with the same "unique identifier" don't have to be identical, and of course there can be many copies of an EPUB file with a given timestamp—we have neatly addressed the needs of the publishers and the technologists, and in a way that is easy for anybody to do.

Here's an example of how the metadata looks in an EPUB:

```
<metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
   <dc:identifier id="pub-id">
      urn:uuid:A1B0D67E-2E81-4DF5-9E67-A64CBE366809
   </dc:identifier>
   <meta property="dcterms:modified">2011-01-01T12:00:00Z</meta>
   ...
</metadata>
```

This results in the Package ID that only a computer could love:

```
urn:uuid:A1B0D67E-2E81-4DF5-9E67-A64CBE366809@2011-01-01T12:00:00Z
```

Although there is an `id` provided for the `dc:identifier` element, it isn't referenced by the `meta` element with the `dcterms:modified` attribute. That's because this `meta` does not "refine" the `dc:identifier`; rather, it applies to the package and is thus a *primary expression*.

Types of Titles

It might also be assumed that one title for an EPUB should be sufficient, and usually, this is in fact the case. However, the EPUB 3 Working Group realized that there are actually quite a few different types of titles that publishers might want to provide in an EPUB's metadata, and that some titles are actually quite complex, with different components serving different purposes. Moreover, different types of publications use different types of titles. ONIX, for example, provides an extensive list of title types used in books; PRISM, the standard for magazine metadata, uses a different scheme.

In keeping with its desire to be both comprehensive, accommodating whatever vocabularies a given publisher might need, as well as being simple to implement and practical as a requirement for reading systems, the EPUB 3.0 specification provides a simple set of six built-in types that reading systems are required to recognize as values of the `title-type` property, but also permits other values with the use of the `scheme` attribute in order to specify where they are documented (e.g., the ONIX Code List 15).

The six basic values of the `title-type` property specified by EPUB 3 are:

main
: The title that reading systems should normally display, for example in a user's library or bookshelf. If no values for the `title-type` property are provided, it is assumed that the first or only `dc:title` should be considered the "main title."

subtitle
: A secondary title that augments the main title but is separate from it.

short
: A shortened version of the main title, often used when referring to a book with a long title (for example, "Huck Finn" for *The Adventures of Huckleberry Finn*) or a brief expression by which a book is known (for example, "Strunk and White" for *The Elements of Style* or "Fowler" for *A Dictionary of Modern English Usage*).

collection
: A title given to a set (either finite or ongoing) to which the given publication is a member. This can be a "series title," when the publications are in a specific sequence (e.g., *The Lord of the Rings*), or one in which the members are not necessarily in a particular order (e.g., "Columbia Studies in South Asian Art").

edition
: A designation that indicates substantive changes from one to the next.

extended
: A fully expressed title that may be a combination of some of the other title types, for example: *The Great Cookbooks of the World: Mon premier guide de caisson, un Mémoire. The New French Cuisine Masters, Volume Two. Special Anniversary Edition*.

The use of these title types provides a good example of how the `id` attribute and the all-powerful `meta` element with its `refines` and `property` attributes are used in EPUB metadata. This is one of many examples provided in the formal EPUB Publications 3.0 specification:

```
<metadata xmlns:dc="http://purl.org/dc/elements/1.1/">
   ...
   <dc:title id="t1">A Dictionary of Modern English Usage</dc:title>
   <meta refines="#t1" property="title-type">main</meta>
   <dc:title id="t2">First Edition</dc:title>
   <meta refines="#t2" property="title-type">edition</meta>
   <dc:title id="t3">Fowler's</dc:title>
   <meta refines="#t3" property="title-type">short</meta>
   ...
</metadata>
```

Note that the other *default vocabulary* values for `property` on `meta`, besides `title-type`, can also be used. Two that publishers may find particularly useful in the context of titles are:

`display-seq`
: Indicates the sequence in which the given `dc:title` should be displayed in relation to the other `dc:titles`

`file-as`
: Provides a version of a title that will alphabetize properly, for example "Canterbury Tales, The."

The Manifest and Spine

The next elements in the EPUB `package` following `metadata` are the `manifest` and `spine`; both of them are required. They constitute the "packing list" and a key aspect of the "assembly instructions" that make an EPUB so much more than just a "website in a box." The `manifest` documents all of the individual resources that together constitute the EPUB, and the `spine` provides a default reading order by which those resources may be presented to a user.

The manifest and Fallbacks

Each and every resource that is part of the EPUB—every content document, every image, every video and audio file, every font, every style sheet: *every* individual resource—is documented by an `item` element in the `manifest`. The purpose is to alert a reading system, up front, about everything it must find in the publication, what kind of media each thing is, and where it can find it. They can be in any order, but they all have to be in the `manifest`.

Each `item` contains these three required attributes:

`id`
: This is essential, so that each constituent part of the EPUB can be uniquely identified. It is what enables that versatile `meta` element to provide metadata associated with specific `item`s, via its `refines` attribute.

`href`
: An internationalized resource identifier (IRI) specifying the location of the resource. Resource names should be restricted to the ASCII character set.

`media-type`
: The MIME media type that specifies the type and format of the resource.

Here's a typical entry for an XHTML content document:

```
<item id="chapter01"
      href="xhtml/c01.xhtml"
      media-type="application/xhtml+xml"/>
```

The EPUB 3 specification designates 14 Core Media Types that all EPUB 3 reading systems are required to recognize. For example, XHTML and SVG are the Core Media Types for content documents; GIF, JPEG, PNG, and SVG are the Core Media Types for images.

Core Media Types are discussed in detail in the other chapters in this book to which they apply.

If a resource in an EPUB is of any *other* media type (which are referred to as *Foreign Resources*), a *fallback* must be provided that *is* a Core Media Type. This is done via the `fallback` attribute on the `item`. Note that an EPUB can have a *fallback chain* in which one non-Core Media Type `item` falls back to another, which eventually resolves to an `item` that is a Core Media Type. We'll look at fallbacks in more detail in "Fallback Content" (page 102) in Chapter 3.

An `item` may also have a `properties` attribute with one or more space-separated property values that alert the reading system to some useful information about the `item`. The values for manifest property values in EPUB 3 are:

`cover-image`
: Clearly documenting which `item` should be displayed as the cover.

`mathml`, `scripted`, `svg`, `remote-resources`, *and* `switch`
: Alerting the reading system to where it will have to deal with MathML, JavaScript, SVG, remote resources, or non-EPUB XML fragments (see the other chapters for more detail on these).

`nav`
: The XHTML5 document that is the required Navigation Document (see Chapter 2).

And finally, an `item` may have a `media-overlay` attribute, the value of which is an IDREF of the Media Overlay Document for the `item`. Media Overlays are the EPUB 3 mechanism for synchronizing text and recorded audio; they are discussed in detail in Chapter 6.

The spine

Whereas the `manifest` documents each and every `item` in the EPUB, in no particular order, the `spine` provides a default reading order, and it is required to list only those components that are not referenced by other components (*primary* content). The point is to provide at least one path by which everything in the EPUB will be presented to the reader, in at least one logical order. So, for example, the `spine` will list the content document for chapter 5 in a book (presumably between those for chapters 4 and 6), but it will not necessarily list images or media or scripts that are contained in chapter 5 (*auxiliary* content), because they will be presented in the context of chapter 5 being presented, and may not necessarily be presented in linear order (e.g., perhaps as a pop up):

```
<spine>
   ...
   <itemref idref="chapter04"/>
   <itemref idref="chapter05"/>
   <itemref idref="chapter06"/>
   ...
</spine>
```

Note that the `spine` is also not the same as the Navigation Document. That document can provide much richer information about the internal structure of content documents (see Chapter 2). The `spine` ensures that a reading system can recognize the first primary content document in the EPUB and then render the successive primary content documents. It does not, however, mandate that a user must access the content in this order; it just provides a *default reading order* that the user is free to depart from.

Instead of the `item` element used in the `manifest`, each element in the `spine` is an `itemref` element. Each of these elements contains an `idref` attribute identifying the appropriate `item` in the `manifest`, as well as an optional `linear` attribute that specifies whether the item is *primary* (`yes`, the default) or *auxiliary* (`no`), and an optional `properties` attribute that can specify whether the given content document starts on the left or right side of a spread:

```
<spine>
   <itemref idref="cover" linear="no"/>
   <itemref idref="chapter01"/>
   ...
</spine>
```

The `spine` may also contain a `toc` attribute that identifies an NCX file. The NCX is how EPUB 2 specified navigation; it is superseded in EPUB 3 by the Navigation Document (`nav`, an XHTML document, as described in Chapter 2). In order to enable an EPUB 3 to be rendered by an EPUB 2 reading system, it has to include an NCX even though it *also* has to have a `nav` to conform to EPUB 3. Until EPUB 2 reading systems have become obsolete (no sign of that in the near future), publishers generally need to include both. The collision is prevented by the fact that the `nav` is pointed to by the `manifest`'s `nav` property and the NCX is pointed to by the `spine`'s `toc` attribute:

```
<package ...>
   ...
   <manifest>
      ...
      <item id="nav"
            href="nav.xhtml"
            media-type="application/xhtml+xml"
            properties="nav"/>
      <item href="toc.ncx"
            id="ncx"
            media-type="application/x-dtbncx+xml"/>
      ...
   </manifest>
   <spine toc="ncx">
      ...
   </spine>
</package>
```

Document Metadata

While the metadata for most EPUBs is typically associated with the publication as a whole, EPUB 3 also enables metadata to be associated with content documents and even with elements within content documents. This is particularly useful for publications that are article-based, like magazines and journals, as well as for *contributed volumes*, books in which each chapter is by a different author. In these cases, much metadata is at the article or chapter level. But you don't need to stop there; you can even associate metadata with elements within the XHTML content documents, down to the phrase level with the `span` element.

It will be no surprise, if you've been paying attention, that this is done with our all-purpose `meta` element. And it depends on each document or element that you want to reference having an `id` attribute that is unique within the EPUB. (Because EPUBs are designed to be self-contained, the IDs are not required to be unique between EPUBs, although for publishers of potentially related EPUBs it is a good practice to do that, to better enable linking and referencing between EPUBs, though most reading systems don't yet enable that.)

Again, the mechanism is a simple and generic one: each `meta` element has a `refines` attribute, the value of which is the ID of what it's referencing, and a `property` attribute that provides either a property value from the default vocabulary (see "The All-Powerful meta Element" (page 5) earlier in this chapter) or a value that uses a prefix and term from one of the reserved vocabularies or from one for which you've declared the prefix.

These `meta` elements are in the `metadata` within the `package`; they are not in the content documents themselves. They reference the content documents (or elements within them) via the relative URI, which means they begin with a # symbol.

Here's an example of some metadata for a hypothetical article from *Sports Illustrated* contained in an EPUB along with other articles, ads, and other resources:

```
<meta refines="#ID12345" property="prism:contentType">article</meta>
<meta property="dc:title" id="t4main">
   Meet The Rejuvenated, Revitalized LeBron
</meta>
<meta refines="#t4main" property="title-type">main</meta>
<meta property="dc:title" id="t4sub">
   After a tumultuous first year in Miami, LeBron James locked himself
   in his house, rued disappointing his teammates—then worked hard to
   hone his game. The result: one of the best seasons in NBA history
</meta>
<meta refines="#t4sub" property="title-type">subtitle</meta>
```

```
    <meta refines="#ID12345" property="dc:creator">LEE JENKINS</meta>
    <meta refines="#ID12345" property="prism:genre">coverStory</meta>
    <meta refines="#ID12345" property="prism:genre">profile</meta>
    <meta refines="#ID12345" property="TimeInc:enhancer">Digitization Co.</meta>
    <meta refines="#ID12345" property="TimeInc:checker">Michael Smith</meta>
```

This metadata all applies to the article that has an ID in the form `id="ID12345"` (note that as a *type ID*, XML rules require this ID to begin with an alpha character). Because the `refines` attribute for `meta` is a relative URI, it begins with a `#` and then the ID of the article.

This example shows that Time, Inc., chose to provide the `contentType` (`article`) and genre categorizations (`coverStory` and `profile`) from the PRISM vocabulary. And they've recorded the firm that created the EPUB and the person who checked it using their own proprietary vocabulary. The `PRISM` and `TimeInc` prefixes would be declared using the `prefix` attribute on the root-level `package` element.

This example also shows the use of the EPUB 3 `title-type` properties for the main title and the subtitle.

Think of how cumbersome it would be if the title and subtitle were combined into one long title!

Links and Bindings

While its main purpose is to provide a specification for consistent, predictable content in EPUBs, the EPUB 3 spec recognizes that it is sometimes necessary to provide things that are not covered by that specification. One way this is done is via the `epub:switch` element, which applies to XML fragments in content documents; it is discussed in detail in Chapter 3. But two other mechanisms are part of the package metadata: the `link` and `bindings` elements.

The OPF meta elements are only intended to be used for identity and version information, and metadata that reading systems might want to expose to users or use to organize and manage the bookshelf. Publishers have much richer "real" bibliographical records, which can be incorporated into an EPUB using a link. Using the `link` element, a child of `metadata` within `package`, is not the same as simply linking to an external location with a link in a content document. The things linked to on the Web from the content documents are not considered part of the publication; and these links do not work if the EPUB is offline. In contrast, the `link` element in an EPUB's `metadata` provides access to a resource that is considered more integral to the publication, although it is not formally part of the EPUB itself. The resource may be provided in the container, so it is available when the EPUB is offline, but a reading system is not required to process or use the resource.

The `link` element requires an `href` attribute to provide either an absolute or relative IRI to a resource, and a `rel` attribute to provide the property value—i.e., what kind of resource is being linked to. The values defined for the `rel` attribute in EPUB 3.0 are:

`marc21xml-record`
: For a MARC21 record providing bibliographic metadata for the publication

`mods-record`
: For a MODS record of the publication conforming to the Library of Congress's Metadata Object Description Schema

`onix-record`
: For an ONIX record providing book supply chain metadata for the publication conforming to EDItEUR's ONIX for Books specification

`xml-signature`
: For an XML Signature applying to the publication or an associated property conforming to the W3C's XML Signature specification

`xmp-record`
: For an XMP record conforming to the ISO Extensible Metadata Platform that applies to the *publication* (not just a component, like an image, for which the prefix mechanism and `meta` element should be used to provide metadata using the `xmp:` prefix)

Here is how you might reference an external ONIX record:

```
<link rel="onix-record" href="http://example.org/meta/records/onix/121099"/>
```

The `bindings` element is a child of the root `package` element. Its chief purpose is to enable an EPUB to contain fallbacks that are more sophisticated than those provided by the HTML5 `object` element's fallback mechanisms. The `bindings` element documents the presence of handlers for such *foreign media* via its `media-type` attribute. When a reading system encounters such an unsupported media type, it looks in the `bindings` element to see if a handler is provided for it, and if so, it is supposed to use that handler instead of the usual fallback. Bindings are discussed in more detail in "Bindings" (page 112) in Chapter 3.

Metadata for Fixed Layout Publications

After the original EPUB 3.0 specification was published in September 2011, it was recognized that despite the importance of EPUB as a reflowable format, many publishers need to use it in a "fixed layout" form in which pagination is fixed, typically via fixed-layout XHTML, SVG, or bitmap images. While there is nothing in that EPUB 3.0 specification to prevent this, it became clear that some metadata should be added to aid in the publication of fixed layout EPUBs. So in March 2012, the IDPF published an informational document to do that.

This provides a number of useful properties. These require the `rendition` prefix to be declared as `prefix="rendition: `*`http://www.idpf.org/vocab/rendition`*`#"` on the `package` element. With one exception as noted below, these properties may be used on the `meta` element to apply to the publication as a whole or on an `itemref` element in the `spine`. The available properties are:

`rendition:layout`
: With the values `reflowable` or `pre-paginated`.

`rendition:orientation`
: With the values `landscape`, `portrait`, or `auto`.

`rendition:spread`
: Specifies how reading systems should render spreads, with the values `none`, `landscape`, `portrait`, `both`, or `auto`.

`rendition:page-spread-center`
: Complements the already existing `page-spread-left` and `page-spread-right` properties to force a specific placement on a spread. It is used only on the `itemref` element in the `spine`.

We'll return to look at how to use these properties to create fixed layouts in more detail in "Fixed Layouts" (page 80) in Chapter 3.

The Container

Now for the easiest part of all: zipping everything up to make it an *.epub*. The specification for this is the Open Container Format (OCF) 3.0, and it is literally a *.zip* file, though it uses the *.epub* extension. It contains all the content documents and other resources, including all the metadata that has been described in this chapter.

All of this is contained in a directory called *META-INF*. This *META-INF* directory contains the following files:

A required container.xml file
: Contains a `rootfiles` element with one or more `rootfile` elements, each of which references a publication's package document, the *.opf* file. Reading systems must use the `manifest` in this `package` (see "The manifest and Fallbacks" (page 16)) to process the EPUB.

> There is an optional *manifest.xml* file that may provide a manifest for the container; this is not used in rendering the EPUB that the container contains.

An optional encryption.xml file
: Holds all the encryption information (if any). Its root element is `encryption`, which contains child elements `EncryptedKey` and `EncryptedData` that describe how the files are encrypted and provide the key to access them. OCF uses the XML Encryption Syntax and Processing Version 1.1 specification. Note, however, that the *META-INF* files themselves as listed here, along with the `package` document that is the root file of the EPUB, must not be encrypted.

An optional manifest.xml file
: Simply provides a manifest for the container (not to be confused with the `manifest` in the EPUB `package`).

An optional metadata.xml file
: May provide metadata for the container (not to be confused with the `metadata` in the EPUB `package`).

An optional rights.xml file
: Contains rights or DRM (Digital Rights Management) information about the EPUB. No DRM scheme is specified; EPUB is deliberately agnostic as to the issue of DRM.

An optional signatures.xml file
: Can hold digital signatures of the container and its contents. Its root element is `signatures`, which contains child `Signature` elements that conform to the XML Signature Syntax and Processing Version 1.1 specification.

The EPUB Open Container Format 3.0 specification provides detailed instructions regarding the creation of the OCF. But for most publishers, this is the simplest part of the process, and is usually quite automated. The result is a single-file EPUB with the extension *.epub* that contains a publication with all the rich content and functionality that EPUB 3 provides—and for which the *.opf* document that is the subject of this chapter provides such an invaluable guide.

One requirement that often trips people up when manually creating an EPUB is that the `mimetype` file has to be the first file added to the ZIP container. The default zipping of an EPUB directory typically results in the *META-INF* directory being packaged first, leading to validation errors. See "Validating unpacked EPUBs" (page 309) in Chapter 11 for information on how to use epubcheck to automate the packaging process and avoid this problem. Free Mac scripts to zip and unzip archives are also available at *http://code.google.com/p/epub-applescripts/*.