Website to Epub Conversion Design

Alexander Rich-Shea

Version 1: April 2, 2012

Introduction

This document is intended to outline a prospective design for software that will convert a Grinnell Computer Science course website into the epub document format.

Program Design

This software is designed to translate one type of file structure to another. A course website has certain basic characteristics that can be generalized to a certain degree. In addition, the structure of an epub is such that conversion from a website format to an epub format is mechanical in terms of file structure of either format. A graphical representation of the two structures is located in Appendix A.

The main functions of the program are therefore involved in the business of creating the epub file structure from the initial website file structure and creating the four main files that include the epib specific functionality:

mimetype
META-INF/container.xml
OPS/content.opf
toc.ncx

The mimetype is an ascii file consisting of a single line of text.
The container.xml file is used to specify the location of the content.opf file.
The content.opf file has three sections:
metadata - author, title, publisher, year, etc.
manifest - a list of all content files
spine - the linear reading order of the ebook.
The toc.ncx file describes the table of content of the epub

We therefore already have many tasks. Right now we will assume that the input to the program is simply the location of a copy of the website files.
1) Build an internal representation of the website directory structure. This is the inital directory structure shown above.
2) Create another directory structure to represent the epub.
    a) create the META-INF and OPS folders.
    b) create the four epub specific files. They are empty for now.
    c) attach the initial directory structure to the OPS folder.
2) write the mimetype. It is the same every time
3) write the container.xml file. Since we know where the content.opf file is located, we can write that information to the container.xml file. This also will be the same every time.
4)write the content.opf file:
    a) write the metadata

- there is no standard source of this information in most course websites.

b) write the manifest

- we can generate a list of files by scanning the initial directory structure.

c) write the spine

-websites are nonlinear structures and therefore there is no way to systematically create a linear structure of the content files that will not be arbitrary to some extent.

5)write the toc.ncx file

-since there is no way to specify the linear reading order of a website, it is similarly impossible to specify its table of contents.

We have an obvious deficiency in our design at this point. Our input does not contain sufficient information to construct the necessary epub specific files. This is the essential problem that this software is solving:

A website is a collection of content files connected in a tree file structure. It is navigated with links that may exist between any two web pages.

An epub is a collection of content files similar to a website, and there may be links between any two content files, but an epub also must have a linear reading structure and a table of contents. In addition it must have metadata, specific pieces of information that are associated with all books, but not all websites. These structures contain semantic information that is not systematically accessible from any given website, including the sites being considered for our program, Grinnell College Computer Science course websites.

The solution to our problem is to require the user to provide a special configuration file to the program. There are now two inputs to our program:
1) a copy of the website files
2) a special configuration file that specifies the metadata, linear reading order, and table of contents that the user would like to add to their website in order to convert it to epub format.

A Consideration of Course Websites

I analyzed the structure of three course websites:

Professor John David Stone - Software Design 323
http://www.cs.grinnell.edu/~stone/courses/software-design/
Referred to below as “Software-Design”

Professor Janet Davis - Computer Networks 364
http://www.cs.grinnell.edu/~davisjan/csc/364/2012S/index.html
Referred to below as “Computer-Networks”

Professor Henry Walker - Algorithms and Object Oriented Design 207
http://www.cs.grinnell.edu/~walker/courses/207.sp12/
Referred to below as “Algorithms-OOD”

These websites vary wildly in structure. Algorithms-OOD contains over 200 files, while Computer-Networks contains closer to 30 files. Software-Design has the simplest folder hierarchy, with only two subfolders in the entire website.

Despite the structural differences, I found a semantic structure that was present to some degree in all three websites.

I propose that course websites be seen as having the following linear structure:
Front Page
Mechanics Page(s)
Schedule Page
Time Unit page(s)
Assignment1s
Assignment2s
References

Assignment types include: labs, supplemental problems, essays
Time of Unit pages include: week, day, none

This structure will guide the formatting of the configuration file.

Input file Structure

The input file is an xml file that specifies the linear order of the epub. This linear order is used to create the spine, the manifest, and the table of contents.

```
<!-- for metadata requirements →
        <title value=”Program Design” />
        <author value=”Alexander Rich-Shea” />
        <language value=”English” />
<!-- The toc tag lets the creator specify the depth of files displayed in the table of contents -->
        <!-- a depth of -1 creates a table of contents with the maximum depth -->
        <toc depth=”2” />

<frontPage location=”index.html” />
</frontPage>
<schedule>
        <file location=”schedule.html” />
</schedule>
<mechanics>
        <!-- an empty directory tag means all files in the order specified-->
                <!-- order values: {char, num, asis} →

        <directory location=”mechanics order=”char”></directory>
        <directory location=”random” order=”asis”>
                <file location=”random/grading.html” />
                <file location=”random/grading2.html” />
        </directory>
</mechanics>
<timeUnits>
        <directory location=”weeks” order=”char” ></directory>
</timeUnits>
<assignment>
        <directory location=”labs” order=”char” > </directory>
</assignment>
<assignment> <!-- more than one assignment tag is allowed -->
        <directory location=”sup-probs” order=”num”></directory>
</assignment>
<reference> <!-- empty first level semantic tags are omitted from the epub -->
</reference>
```

A Possible Main Function

```
allocate space for:
      directory_structure   website
      directory_structure   epub
      metadata         metadata
      manifest         manifest
      spine                  spine
      file             mimetype
```

```
      tree            toc
      xml_document    container.xml
      xml_document    content.opf
      xml_document    toc.ncx

//initialize website tree structure
      build_init_dir_structure( website );
//initialize epub tree structure
      build_epub( epub, website );
//parse the config file to initialize the metadata, spine, and toc
      parse_config_file( metadata, spine, toc );
//initialize the mimetype file
      build_mimetype( mimetype );
//initialize the manifest
      build_manifest ( manifest, website );

/*
with all of the information pulled from the website and     config-
file, we start writing things
*/

write_toc( toc, epub);
write_content_and_container( metadata, manifest, spine, epub );
write_mimetype( mimetype, epub );

zip_and_rename( epub );
free_structures();
```

Possible Structs

The following are possible struct definitions for our custom data types listed above. It remains to be solved as to how we will represent the directory_structure type.

The metadata struct contains the 3 requires pieces of metadata as well as the author metadatum.
The two node structures contain variables named according to their tag's required attributes in the content.opf xml file.
The spine and manifest structures act as head nodes for linked lists that represent the spine and manifest collections.

```
struct {

  char * title;
  char * language;
  char * identifier;
  char * author;

} metadata;
```

```
struct {

   char * id;
   char * href;
   char * media-type;
   itemNode * next;

} itemNode;
```

```
struct {

   int count;
   itemNode * next;

} manifest;
```

```
struct {

   char * idref;
   char * linear;
   itemRefNode * next;

} itemRefNode;
```

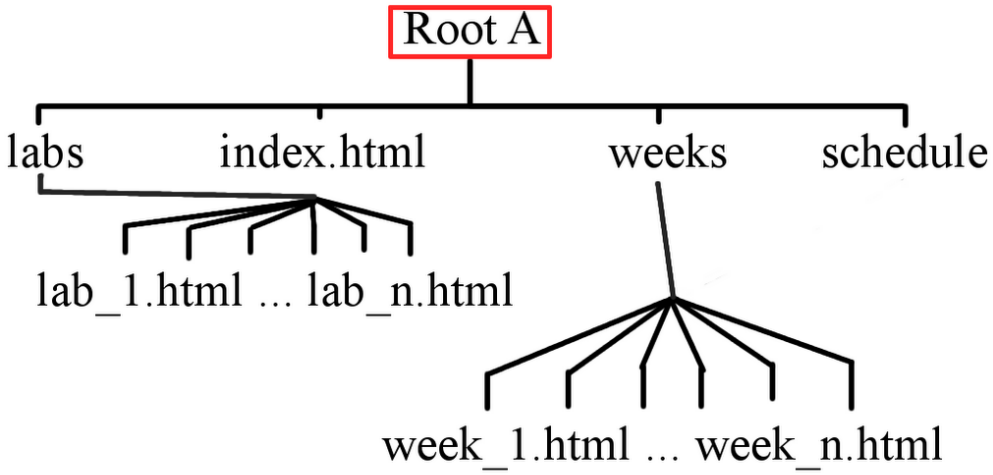
```
struct {

   int count;
   char * toc;
   itemRefNode * next;

} spine;
```

External Libraries

libxml2 - for writing xml files

Appendix A

Initial Directory Structure



Epub Directory Structure