Table of contents

Preface

This page is part of the took titled OpenStax Publishing with a WYSIWYG Editor.

The elusive WYSIWYG CNXML editor

Have you ever wished that you could use a WYSIWYG editor to create and publish CNXML content on OpenStax. Well, that is what I was wishing for in 2009. Rather than

just wish, however, I did something about it. I wrote an XHTML-to-CNXML translator program for translating XHTML files into CNXML files suitable for uploading and publishing on OpenStax. *(OpenStax was called Connections at that point in time.)*

If you go to the Advanced Search feature at OpenStax CNX and search for an author named Richard Baldwin, you will learn that I have used my translator program along with Microsoft Expression Web 4 to create and publish more than 770 Books and Pages on OpenStax since 2009. My approach really does work and it is easy to use.

How I create and publish CNXML content on OpenStax

These are the four steps that I use to create and publish CNXML content on OpenStax:

1. For each new Page, I use the free version of Microsoft Expression Web 4 to create a valid XHTML file describing my content. This mostly involves the use of the built-in WYSIWYG editor in Expression Web 4.
2. Then I process the XHTML file through my XHTML-to-CNXML translator program named **CNXMLprep12** to produce a CNXML file suitable for uploading to OpenStax.
3. Then I upload the CNXML file to OpenStax.
4. Last but not least, I publish the CNXML file on OpenStax.

It couldn't be simpler. I do everything at the XHTML level, mostly using the WYSIWYG editor. I rarely touch the raw XHTML code and I never touch the raw CNXML code.

I maintain all of my archives in XHTML format. When I need to update a page, *(as I am doing right now)*, I update the corresponding XHTML file, run it through my translator, upload the modified CNXML file to OpenStax, and publish it. Once again, I never touch the raw CNXML code either before or after uploading it to OpenStax.

You too can publish CNXML content on OpenStax

In support of my strong interest in OER and free textbooks, I have decided to make my translator program available for use by the general public.

If you don't know anything about CNXML but you already know how to create a valid XHTML file using a WYSIWYG editor such as the free version of Microsoft Expression Web 4, you can easily create and publish your CNXML content on OpenStax using my free XHTML-to-CNXML translator program.

If you don't know about XHTML but you do already know how to use Microsoft Word, you should have no difficulty learning how to use Microsoft Expression Web 4. That is because the user interface of Microsoft Expression Web 4 is very similar to the user interface of earlier generations of Microsoft Word.

By now you might be asking, *"Why not just create and upload Microsoft Word documents to OpenStax?"* Click here to learn my reasons for not doing that.

This page explains how to use my translator program to create and publish your content. This page also provides a link to a downloadable zip file that contains everything you need to get started *(including executable Java program files and an XHTML template file)*. The one thing that the zip file doesn't include is a WYSIWYG XHTML editor. However, as of March 2016, you can download the free version of Microsoft Expression Web 4.

While you should be able to use any WYSIWYG XHTML editor, I recommend that you use Microsoft Expression Web 4 if possible because it is free, because it has a built-in validator, and because it is relatively easy to use.

By the way, this page was produced using the free version of Microsoft Expression Web 4 in conjunction with my translator program.

Viewing tip

I recommend that you open another copy of this module in a separate browser window and use the following links to easily find and view the Figures and Listings while you are reading about them.

Figures

Listings

Discussion

My translator program is named **CNXMLprep12**. As you can see from the number at the end of the name, it has gone through quite a few improvements and iterations since 2009.

How it works

This section contains a brief description of how the program works. If you don't care how it works, feel free to skip to the next section. If you are not interested in any

background information at this point and you just want to create, upload and publish some content on OpenStax, skip ahead to the section titled Run the program.

The program consists of three major Java classes:

- CNXMLprep12
- CNXMLFirstPass12
- CNXMLSecondPass12

The first class in the list is the driver.

Code in an object of the class named **CNXMLFirstPass12** reads a valid XHTML file and performs several cleanup actions to make it easier to convert the XHTML file to CNXML format. It writes a temporary output file named **CNXMLtemp.html** in XHTML format in the current folder.

Following that, code in an object of the class named **CNXMLSecondPass12** reads the file named **CNXMLtemp.html** and translates it into CNXML code suitable for uploading to OpenStax. *(See the section named Run the Program for instructions on how to create and upload a CNXML file to OpenStax.)*

Instructions regarding your XHTML code

I assume that you will be creating content containing headers, ordinary paragraph text, preformatted text, ordered lists, unordered lists, blockquote, tables, images, and programming source code, along with other kinds of structures commonly found on web pages. This section contains guidelines on how best to create that content in your XHTML editor.

Before going further, I need to point out that there are many CNXML tags that cannot be created by this program. This is not a complete WYSIWYG CNXML editor. Rather, it is a program designed to support a reasonable subset of approximately 57 tags that are available in CNXML.

Preformatted text and program source code

If you include program source code *(such as Java or C++ source code)* in the XHTML document, it must be placed in a preformatted block. For example, the code shown in Listing 1 was placed in a preformatted block in a table. It was placed in the table to make it easy to provide a caption and an anchor.

Listing 1. Preformatted text.

```
out.println("<html xmlns=\"http://www.w3.org" +
                              "/1999/xhtml\">");
```

Listing 1. Preformatted text.

```
out.println("<head>");
out.println("<meta content=\"text/html; " +
"charset=utf-8\" http-equiv=\"Content-Type\"/>");

out.println("<title>dummy title</title>");
out.println("</head>");
out.println("<body>");<out.println("<body>");
```

There must not be any formatting code, such as **bold text** inside a preformatted block.

All left angle brackets, right angle brackets, quotes, and ampersands in preformatted text must be converted to the following entities in the raw XHTML code before attempting the translation into CNXML.

```
< = &lt;
> = &gt;
" = &quot;
& = &amp;
```

As an alternative to placing preformatted text in a table, free-standing preformatted text such as the source code shown below is also allowed. Just be sure to deal with the angle brackets, quotes, and ampersands as described above if you create free-standing preformatted text.

```
import org.newdawn.slick.AppGameContainer;
import org.newdawn.slick.BasicGame;
import org.newdawn.slick.GameContainer;
import org.newdawn.slick.Graphics;
import org.newdawn.slick.SlickException;

public class Slick0130a extends BasicGame{

  //Instance variables for use in computing and
  // displaying total time since program start and
  // time for each frame.
  double totalTime = 0;
  int incrementalTime = 0;

  public Slick0130a(){
    //Call to superclass constructor is required.
    super("Slick0130a, Baldwin.");
  }//end constructor
  //-----------------------------------------------------//
```

Hyperlinks

The program translates hyperlinks for local anchors *(inside the document)* and for external web sites.

All hyperlink target names must be unique. You must also make certain that target names for local anchors don't include spaces or punctuation characters such as comma, colon, semicolon, dash, etc. Just use letters, numbers, and underscore characters *(Figure_01 for example)* for your local anchor target names.

Table considerations

XHTML tables and CNXML tables come in many different varieties and formats. This program supports a subset of those varieties and formats.

The table "width" attribute

The XHTML table **width** attribute is translated into a CNXML **pgwide** attribute. Here is what the OpenStax documentation has to say about the **pgwide** attribute:

> **pgwide** (optional): Determines the available width of the table.
>
> Possible values:
>
> - **0** - Maximum width of the table is the galley width (default).
> - **any positive integer** - The width of the table is the entire width of the page.

What this really seems to mean is that for a **width** value of 0, the table will only be wide enough to display its contents. For a **width** value of 1, the table will be the full width of the OpenStax page in the browser.

As of March 16, this width variation only works in the Legacy view. When the table is viewed in the newer OpenStax view, all tables seem to be full width regardless of the value of **pgwide**. I assume that this is an error that will be fixed sometime in the future.

In this program, the table **width** value must be either 0 or 1. Percent (%) values are not allowed, nor are any values other than 0 or 1.

The table "border" attribute

Tables in the XHTML file must have **border** attribute values of either 15 or 20. No other values are allowed in this version of the program.

A **border** value of 15 results in a CNXML **note** object being created as shown by the two note objects in the Miscellaneous section.. *(A CNXML **note** object displays as the full width of the browser window regardless of the value of the **width** attribute.)*

A **border** value of 20 results in a CNXML **table** object as shown in Figure 1.

Structure of a simple table

This program requires that all **table** objects have the full structure shown in Figure 1.

Figure 1. Required table structure.

```
table border="15 or 20" width="0 or 1"
  thead
    tr
      th
        Caption text
      /th
    /tr
  /thead

  tbody
    tr
      td
        Table content
      /td
    /tr
  /tbody
/table
```

The easiest way to create such a table is probably to copy a table from the template that I will provide, paste it into your WYSIWYG editor, and then edit as needed. However, you should be able to create an acceptable table using only the WYSIWYG features of your XHTML editor if you prefer not to use a template.

Structure of multi-column, multi-row tables

I normally place my Figures and Listings in tables having a header, one column, and one row. I provide an anchor and a caption in the header. Therefore, since I publish a lot Figures and Listings, I use a lot of tables of the sort shown in Figure 1, Figure 3, and Listing 1

Only occasionally do I need a table with multiple columns and multiple rows. The template that I will provide has examples of several different styles of multi-column, multi-row tables. This is definitely a case where it is probably easier to copy, paste, and edit than to start from scratch and build the table.

It is worth noting that if a single header is created in a multi-column table, it normally appears above the first column with a width that is limited by the width of the column.

Ordinary text

All ordinary text must be in a paragraph or some other suitable element such as a table or a blockquote. You won't be able to upload your content to OpenStax if you create text in your XHTML document that isn't properly contained.

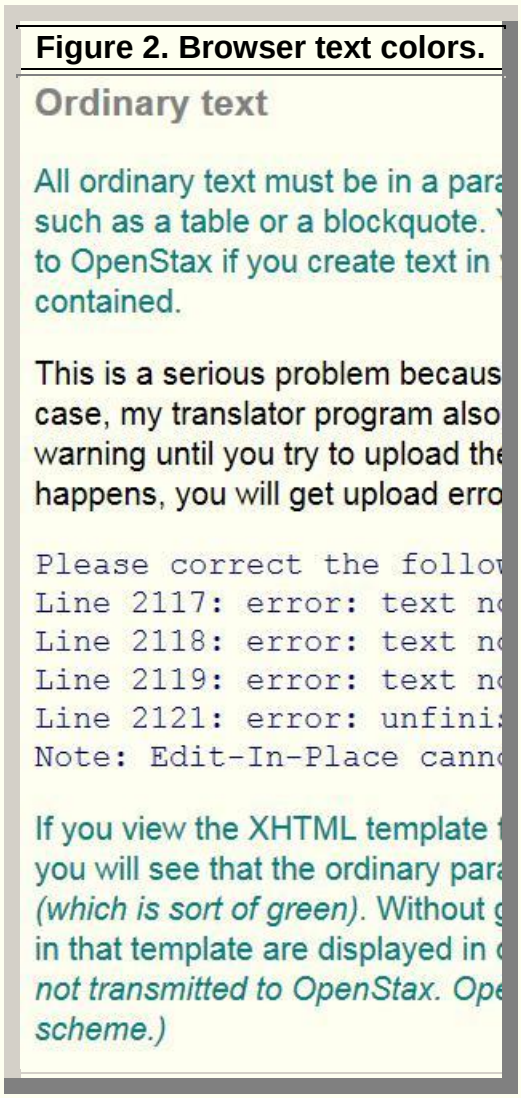
This is a serious problem because your validator may not catch it. In that case, my translator program also won't catch it and you won't get any warning until you try to upload the CNXML file to OpenStax. If that happens, you will get upload errors from OpenStax similar to the following:

```
Please correct the following errors:
Line 2117: error: text not allowed here
Line 2118: error: text not allowed here
Line 2119: error: text not allowed here
Line 2121: error: unfinished element
Note: Edit-In-Place cannot be used while errors persist
```

If you view the XHTML template file that I have provided in your browser, you will see that the ordinary paragraph text is displayed in the color teal *(which is sort of green)* as shown in the first and last paragraphs of Figure 2. Without going into detail, different kinds of content in that template are displayed in different colors. *(The color information isn't transmitted to OpenStax. OpenStax displays everything in its own color scheme.)*

Figure 2. Browser text colors.

Figure 2. Browser text colors.



If you use that template and you create text that is not properly contained in a suitable element, it will be displayed in black, as shown by the second paragraph in Figure 2, when you view the XHTML file in the browser. That usually makes it easy to spot and fix before it creates problems later on.

XHTML headers and CNXML sections

XHTML headers such as **h1**, **h2**, etc., are used as delimiters when creating **sections** in the CNXML code. CNXML sections begin and end at XHTML headers such as **h1**, **h2**, etc.

XHTML headers must be nested in a hierarchical manner. In other words, **h2** must be a child of **h1**, **h3** must be a child of **h2**, etc.

The hierarchical nature of this document is illustrated in the Table of Contents. Each item in the Table of Contents was a link to a header *(such as **h1**, **h2**, **h3**, etc.)* in the XHTML document and is a link to the beginning of a section in this CNXML document.

Extraneous entities

Once you have finished editing your XHTML file, it may contain one or more entities that can't be handled by this program. *(The most common entity is the so-called non-breaking space entity shown below.)*

```
 
```

Some entities *(such as those shown here)* are required while others are not allowed. Any entities that are not allowed will be identified by error messages when you run the program. As far as I know, they all begin with an ampersand and end with a semicolon as shown above.

The bad entities must be removed from the XHTML file before it can be successfully processed by this program. This is one place where your WYSIWYG editor will fail you. Probably the best way to remove entities is to switch to the code-editor window of your editor program, search out the bad entities, and delete them. Just be sure not to delete any of the good entities in the process.

Images

Free-standing images such as the one shown below are allowed by the program.

However, if like me you often need to put captions and anchors on your images, such as those shown in Figure 3, you can put each image in a table with a **border** value of 20 and a **width** value of 0. Put the caption and the anchor in the table header.

Figure 3. Image in a table.

Figure 3. Image in a table.

All **img** elements must be closed with the / character. Some editors *(including the free version of Microsoft Expression Web 4 which I use)* don't do that. Therefore, it is necessary for me to edit the raw XHTML code to add those / characters to the **img** elements.

Validating your XHTML file

Validate your finished XHTML file as *XHTML 1.0 Transitional* using any good XHTML validator. The free version of Microsoft Expression Web 4 has a built-in validator and it is not necessary to put a DTD declaration in your XHTML file.

Some validation software may require that you include a DTD declaration at the top of your XHTML file in order to validate the file. If that is the case with your validator, you MUST remove the DTD declaration from the XHTML file before using the file as input to this program.

A DTD declaration normally looks something like the following, beginning with **!DOCTYPE** and ending with **.dtd** and surrounded by angle brackets.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

If you don't want to physically remove it from your XHTML file, you can temporarily disable it by turning it into a comment as shown below:

```
<!--
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
-->
```

Don't use XHTML break tags

Break tags are used in XHTML to create a series of lines of text with no space in between. An XHTML break tag looks like this:

```
<br/>
```

To my knowledge, there is no CNXML tag that maps into an XHTML break tag. Therefore, there must not be any XHTML break tags in the temporary output file produced by an object of the class named **CNXMLFirstPass12**. The behavior of an object of that class is to replace all XHTML break tags with XHTML paragraph tags. Although this works in some cases, it can also cause "well formed" errors later downstream in other cases.

The best approach is to manually remove all XHTML break tags from your finished XHTML file before processing it with this program. In other words, DON'T USE XHTML break tags in your XHTML file unless it is absolutely necessary.

CNXML editing is not required

It should not be necessary for you to manually edit the CNXML file produced by this program either before or after uploading it to OpenStax. If there are problems with the upload of the CNXML file, or problems with the published result, it should be possible for you to make any necessary corrections at the XHTML level. Then re-run this program to create a new CNXML file for uploading. I have been doing that since 2009 and with more than 770 books and pages published on OpenStax, I have never found it necessary to edit CNXML code.

No CNXML figure objects or CNXML listing objects

Early versions of this program supported CNXML **figure** objects and CNXML **listing** objects. Although those objects sound good in theory, I have encountered significant display problems with objects of those types, particularly when the site moved from the Legacy presentation format to the OpenStax presentation format. As a result of those problems, I had to go back and remove those objects from most of my pages. Following that, I updated the program to disallow the creation of CNXML code for **figure** objects or **listing** objects.

My approach, which works very well for me, is to put figures and listings into CNXML **table** objects and to distinguish between figures and listings in the captions, which I place in the table header. *(See Figure 3 and Listing 1 for example.)*

Run the program

This section explains how to run the program and how to publish your CNXML content on OpenStax.

Running the program

This program requires access to the following three compiled Java class files plus a valid XHTML file for input

- CNXMLprep12.class
- CNXMLFirstPass12.class
- CNXMLSecondPass12.class

Click here to download a zip file named **CNXMLprep12.zip** containing those three class files. Click here to download a zip file named **CNXMLtemplate01.zip** containing a valid XHTML template file named **CNXMLtemplate01.htm** that illustrates many of the capabilities of this program. The zip file also contains several image files associated with the template file.

To run this program in general, extract the contents of the zip file named **CNXMLprep12.zip** into an empty folder, copy your validated XHTML file into that folder, and execute the following command at the command prompt in that folder:

```
java CNXMLprep12 InputFile OutputFile
```

where **InputFile** is the name of your validated XHTML file and **OutputFile** is the name of the CNXML file that the program will produce.

For this to work, you must have the Java runtime engine *(JRE)* installed on your computer. The JRE is already installed on many computers and it may already be installed on yours. If not, there are many web pages that provide instructions for downloading and installing Java. One of those web pages is located here.

To run this program with the template file named **CNXMLtemplate01.htm**, extract the contents of both zip files into an empty folder. Then execute the following command at the command prompt in that folder:

```
java CNXMLprep12 CNXMLtemplate01.htm CNXMLtemplate01.cnxml
```

Two new files should appear in the folder:

- CNXMLtemp12.html
- CNXMLtemplate01.cnxml

You can ignore the temp file. It is of no further use.

The file named **CNXMLtemplate01.cnxml** is the output CNXML file produced by the program.

Upload and publish your new page

Here is an abbreviated description of the steps for uploading and publishing your new page.

1. Click here to create an OpenStax account and sign in to your new account. *(See helpful hints below)*

2. Create a new **workgroup** or use the workgroup named **Personal Workspace** to create a new **module**.

> **Historical context**: In the days when the website was known as **Connexions**, before it became known as **OpenStax**, individual CNXML files were known as **Modules** and a collection of such files was known as a **Collection**. With the advent of the **OpenStax** name, individual CNXML files became known as **Pages**, and collections of such files became known as **Books**. Much of the old terminology still appears on the website, particularly in those areas of the site associated with creating and publishing content. Therefore, you will be creating a **Module**, which will be known as a **Page** once it is published and viewed in the **OpenStax** format. However, if you view it in the **Legacy** format *(link on the upper-right corner of the **Page** screen)*, it will still be known as a **Module**.

3. Enter the Metadata for the new module *(title, keywords, summary, etc.)*

4. Upload the file named **CNXMLtemplate01.cnxml** to OpenStax as type ***Plain CNXML***.

5. Put the five image files that you downloaded earlier *(**0135ex02.jpg** through **0135ex06.jpg**)* in a zip file and upload it to OpenStax as type ***Zip File***.

> **Important Note**: The XHTML code must be written so as to access image files and other resource files from within the same folder as the validated XHTML file. References to resource files in other folders are not allowed in the XHTML code.

6. Finally, follow the OpenStax instructions to publish your new content.

Helpful hints regarding OpenStax authoring

Once you have signed in to OpenStax, go to Authoring Content. That page provides links to several other pages containing helpful hints regarding the authoring of content on OpenStax. The page titled Create a Module might be particularly helpful in describing the steps involved in creating and publishing a module. Just remember, that you already have the CNXML code for your module. You simply need to import and publish the module. *(You will be importing CNXML instead of Microsoft Word.)*

You will also find links to other useful authoring pages on your MyCnx page.

The utf8 character set

The XHTML input file must be coded as **charset=utf8**. Be careful with this requirement because some XHTML editors create text using other character sets. The safest approach is to start with my XHTML template file that contains a **meta** line in the **head** section that looks something like the following:

```
<...charset=utf-8...>
```

Then edit that template file to meet the needs of your page.

Why not create and upload Microsoft Word documents to OpenStax?

The OpenStax documentation indicates that the following file types can be imported:

- Microsoft Word
- OpenOffice Writer
- LaTex
- Plain CNXML
- Zip File

(If you use my XHTML-to-CNXML translator, you will be importing "Plain CNXML" and "Zip File" into OpenStax.)

Now for the primary question posed by this section -- to begin with, when you select Microsoft Word as the import type, you are immediately confronted with the following disclaimer:

Word documents can be extremely varied. We have attempted to handle many common cases, with the goal of extracting the text from your document so that ***you can mark it up in CNXML*** without having to retype the content. Some cautions:

- Images: Many images will be imported without trouble. Images contained in figures or tables, created directly in Word, or embedded in the document through links may not work correctly.
- Styling Tips: If you use CNXML-specific styles from the Word template to create your Word document, you will get much more faithful conversion for elements like terms, citations, code, and others. See the styling information in our full instructions, below.
- Use Headings to Get Sections: Even without CNXML-specific styling, you can make sure that your section organization is preserved during import by using the standard header styles (Heading 1, Heading 2, etc.) in your original document.
- Full instructions on using Word with OpenStax-CNX

That is hardly a confidence builder. If you follow the link to the Full instructions, you will find the following at the beginning of the page.

> **Overview**
>
> One of the easiest paths for populating a module is the Word/OOo importer, which converts a *.doc document to CNXML. The importer overwrites any existing CNXML, so it is most useful as an ***initial import. We suggest using the online Edit-in-Place feature to make any further edits***; any re-import of a Word or OpenOffice document will erase any other changes you have made using Edit-in-Place or the Full Source Editor.
>
> **Who should use the Word/OOo importer?**
>
> You should use the Word/OOo importer if you already have a Word or OpenOffice document saved that you wish to publish to the repository...

It would appear from the above that the primary purpose of the Word importer is to provide a "starting point" CNXML document to be further edited online in CNXML code.

> *(In contrast, if you create your document in WYSIWYG XHTML and use my XHTML-to-CNXML translator to prepare it for upload to OpenStax as type Plain CNXML, you never have to touch or even think about CNXML code. You can handle creation as well as ongoing maintenance at the WYSIWYG XHTML level.)*

Actual results: I opened this XHTML file in Microsoft Word and then saved the file both as a **doc** file and a **docx** file. Then I tried to import both of those files into OpenStax as type Microsoft Word. In both cases, I received an error message that read "**Could not import file. Could not convert file**."

There was no indication as to the cause of the problem nor was there any suggestion as to how to resolve the problem. Therefore, in my case, trying to import a Microsoft Word document into OpenStax has proven to be a dead end street.

That is my reason for not creating and uploading Microsoft Word documents to OpenStax?"

Summary

If you can create a valid XHTML file using a WYSIWYG editor such as the free version of Microsoft Expression Web 4, you can easily create and publish content on OpenStax using my free XHTML-to-CNXML translator program. This page explains how to use my translator program to create and publish your first page on OpenStax.

Miscellaneous

This section contains a variety of miscellaneous information.

Housekeeping material

- Module name: Wysiwyg0100 Getting Started
- File: Wysiwyg0100.htm
- Published: 03/18/16
- Revised: 03/20/16 to remove discussion of footers

Disclaimers:

Financial: Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.

In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation: I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

-end-