PreTeXt Author's Guide

PreTeXt Author's Guide

Robert A. Beezer
University of Puget Sound

DRAFT November 26, 2018 DRAFT

Preface

This guide will help you *author* a PreTeXt document. So it serves as a description of the PreTeXt XML vocabulary, along with the mechanics of creating the source and common output formats. Note that this is different than *publishing* your document, which is described in the *PreTeXt Publisher's Guide*. Even if you intend to distribute your document with an open license, and you are both author *and* publisher, it is still helpful and instructive to understand, and separate, the two different steps and roles.

Contents

Chapter 1

Start Here

Welcome to the Author's Guide for PreTeXt. You are likely eager to get started, but familiarizing yourself with this chapter should save you a lot of time in the long run. We will try to keep it short and at the end of early chapters we will guide you on where to go next. Not everything we say here will make sense on your first reading, so come back after your first few trial runs. When you are ready to seek further help, or ask questions, please read the Welcome to the PreTeXt Community in Appendix A.

1.1 Philosophy

PreTeXt is a **markup language**, which means that you explicitly specify the logical parts of your document and not how these parts should be displayed. This is very liberating for an author, since it frees you to concentrate on capturing your ideas to share with others, leaving the construction of the visual presentation to the software. As an example, you might specify the content of the title of a chapter to be `Further Experiments`, but you will not be concerned if a 36 point sans-serif font in black will be used for this title in the print version of your book, or a CSS class specifying 18 pixel height in blue is used for a title in an online web version of your book. You can just trust that a reasonable choice has been made for displaying a title of a chapter in a way that a reader will recognize it as a name for a chapter. (And if all that talk of fonts was unfamiliar, all the more reason to trust the design to software.)

You are also freed from the technical details of presenting your ideas in the plethora of new formats available as a consequence of the advances in computers (including tablets and smartphones) and networks (global and wireless). Your output "just works" and the software keeps up with technical advances and the introduction of new formats, while you concentrate on the content of your book (or article, or report, or proposal, or . . .).

If you have never used a markup language, it can be unfamiliar at first. Even if you have used a markup language before (such as HTML or basic LaTeX) you will need to make a few adjustments. Most word-processors are WYSIWYG ("what you see is what you get"). That approach is likely very helpful if you are designing the front page of a newspaper, but not if you are writing about the life-cycle of a salamander. In the old days, programs like `troff` and its predecessor, `RUNOFF` (1964), implemented simple markup languages to allow early computers to do limited text-formatting. Sometimes the old ways are the best ways.

PreTeXt is what is called an **XML application** or an **XML vocabulary** (I prefer the latter). Authoring in XML might seem cumbersome at first, but you will eventually appreciate the long-run economies, so keep an open mind. And if you are already familiar with XML, realize we have been very careful to design this vocabulary with human authors foremost in our mind.

Principles. The creation, design, development and maintenance of PreTeXt is guided by the following list of principles. They may not be fully understood on a first reading, but should be useful as you become more familiar with authoring texts with PreTeXt and should amplify some of the previous discussion.

1. PreTeXt is a markup language that captures the structure of textbooks and research papers.
2. PreTeXt is human-readable and human-writable.
3. PreTeXt documents serve as a single source which can be easily converted to multiple other formats, current and future.
4. PreTeXt respects the good design practices which have been developed over the past centuries.
5. PreTeXt makes it easy for authors to implement features which are both common and reasonable.
6. PreTeXt supports online documents which make use of the full capabilities of the Web.
7. PreTeXt output is styled by selecting from a list of available templates, relieving the author of the burden involved in micromanaging the output format.
8. PreTeXt is free: the software is available at no cost, with an open license. The use of PreTeXt does not impose any constraints on documents prepared with the system.
9. PreTeXt is not a closed system: documents can be converted to LaTeX and then developed using standard LaTeX tools.
10. PreTeXt recognizes that scholarly documents involve the interaction of authors, publishers, scholars, instructors, students, and readers, with each group having its own needs and goals.

List 1.1.1: PreTeXt Principles

1.2 Formatting Your Source

There are a lot of details related to how you prepare your **source**: the actual files that you, and you alone, will create. At least skim through the following, come back here often, and also consult (((exhaustive-chapter-on-source-files))).

File Format. Your source should be plain **ASCII files** which you will create with a text editor. In other words, do not create your source with Word, LibreOffice, WordPerfect, AbiWord, Pages or similar programs. Popular text editors include vi, emacs, Notepad, Notepad++, Atom, TextWrangler, and BBEdit. I have had a very good experience with Sublime Text, which is cross-platform (Windows, OS X, Linux), and can be used for free, though it has a very liberal license and is well worth the cost. Sometimes these editors are known as a **programmer's editor** (though we will be doing no programming). Support for writing HTML sometimes translates directly to good support for XML.

There are **XML editors**, which I have generally found too complex for authoring in PreTeXt. They do have some advantages and XML Copy Editor is one that I have found that is possibly useful.

Learn to Use Your Editor. Because XML requires a closing tag for every opening tag, it feels like a lot of typing. Your editor should know what tag to close next and there should be a simple command to do that. Discover this first and consider switching editors if it is not available. For me, in Sublime Text on Linux, I just press `Alt-Period` and get a closing tag. Not only is this quick and easy, I often recognize that I am not getting the tag I expected since I forgot to close one earlier. This one shortcut can pretty much

cut your authoring overhead in half.

If your editor can predict your opening tag, all the better. Sublime Text recognizes that I already have a `<section>` elsewhere, so when I start my second section, I very quickly (and automatically) get a short list of choices as I type, with the one I want at the top of the list, or close to it.

Invest a little time early on to learn, and configure, your editor and you can be even more efficient about capturing your ideas with a minimum of overhead and interference.

Revision Control. If you are writing a book, or if you are collaborating with co-authors, then you owe it to yourself and your co-authors to learn how to use revision control, which works well with PreTeXt. The hands-down favorite is `git` which has a steep learning curve, and so is beyond the scope of this guide. But see (((topic-git-coexistence))) which has hints on how to best use `git` together with a PreTeXt project and look for Beezer and Farmer's *Git For Authors*.

Whitespace. The term **whitespace** refers to characters you type but typically do not see. For us they are **space**, **non-breaking space**, **tab** and **newline** (also known as a "carriage return" and/or "line feed"). Unlike some other markup languages, PreTeXt *does not ever use whitespace* to convey formatting information.

In some parts of a PreTeXt document, every single whitespace character is important and will be transmitted to your output, such as in the `<input>` and `<output>` portions of a `<sage>` element. Since Sage code mostly follows Python syntax, indentation is important and leading spaces must be preserved. But you can indent all of your code to match your XML indentation and the entire `<input>` (or `<output>`) content will be uniformly shifted left to the margin in your final output.

In other parts of a PreTeXt document, every single whitespace character is ignored, and you have the freedom to use indentation and blank lines to help you understand the logical structure of your document. An example is that you can add as much whitespace as you like between the paragraphs of a section, such as a preceding blank line and indentation, and none of it will affect your output in any way.

Never use tabs, they can only cause problems. You may be able to set your editor to translate the tab key to a certain number of spaces, or to translate tabs to spaces when you save a file (and these behaviors are useful). I have Sublime Text configured to show me every single space as a small faint dot, since I like to be certain I have no stray whitespace *anywhere*.

Structure of your Source. XML is hierarchical, like a family tree. Books contain chapters, chapters contain sections, sections contain paragraphs, etc. I like to reflect each new level of containment by consistently indenting four spaces (you might prefer two spaces). A good editor will visually respect this indentation, and help you with maintaining the right indentation with each new line, so much so, that you will forget how much assistance it is providing.

Develop a style and stick with it. I put titles on a new line (indented) after I create a new chapter or section, some people like them on the same line, immediately adjacent. I put a single blank line before each new paragraph, but not after the last. And so on. The choice is yours, but consistency will pay off when you inevitably come back to edit something. You have put a lot of work and effort into your source. You will be rewarded with fewer problems if you keep it neat and tidy, and you will also get very clean output.

1.3 Where Next?

We will end each of the early chapters with suggestions on where to read next.

If you are impatient (sometimes a good quality!), then Chapter 2 should be next, where we construct and process a short example and then expand it slightly with several additions.

If you would like a general, high-level overview of features skip ahead to Chapter 3.

If you have an existing project authored in LaTeX you may be interested in the conversion process described in Appendix D.

Chapter 2

A Careful, Quick, Minimal Example

We are going to walk you through a simple example carefully, with some explanation along the way. There are two steps. First we will author your source and then after some setup, we will process your source file into output.

2.1 Authoring

Having read already about text editors in Section 1.2, fire up your text editor and begin a text file that you will name `quickstart.xml`.

Though it is optional, it will be good practice to always make the first line of your file the following. This will identify your source file as an XML file, but the exact particulars are not important now.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Every PreTeXt document is enclosed inside `<mathbook>` tags and we will be writing an `<article>`. So extend your source to the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<mathbook>
    <article>
    </article>
</mathbook>
```

Notice how every opening tag (no slash) has a paired closing tag (with a slash, plus the identical name). We would like to our article to have some content, so we will add a short paragraph with a single sentence. So finish your file off as follows and save it.

```
<?xml version="1.0" encoding="UTF-8"?>
<mathbook>
    <article>
        <p>This is a short sentence.</p>
    </article>
</mathbook>
```

That's it. You now have a complete, properly-formed PreTeXt document. You are ready to organize the two tools necessary to process your source into different formats.

2.2 Setup

There are two components to processing your document, the PreTeXt stylesheets and the `xsltproc` program. We work at the **command-line** inside of a **terminal** or **console**. If you do not know what this is, it will

seem very primitive at first. Sometimes the old ways are the best ways. This will be called a "Command Prompt" in Windows or a "Terminal" on a Mac. In Linux it may be known as a "console" or a "shell". A tutorial, which is Linux-specific, can be found at Ryan's Tutorials and certainly others exist.

The operating system on a Mac is built on Unix, which is very similar to Linux, so most of the directions here will be little changed between the two. Procedures can be very different in Windows (Appendix H, Appendix I). One alternative is CoCalc which provides a full Linux computer for free in your web browser, so that may be an excellent place for initial experiments (Section 4.12).

Step 1: PreTeXt. You need to obtain the PreTeXt stylesheets, which are the main part of PreTeXt. Since you are reading this, it may be possible that you have this already. You can use `git` to **clone** the PreTeXt from the GitHub repository, and then be sure to checkout the `dev` branch to have the latest version. This is the best way to go, and you should only download the repository as a zip file once for an initial experiment, and then switch to using a clone instead.

Once you have a clone of the repository, you can issue `git pull`, and git will update your local copy with any recent changes. You should do this *regularly*—meaning on the order of *daily*. See the FAQ entry for more about why we *expect* you to do this.

See the website at `mathbook.pugetsound.edu` for details and commands for this step, right on the main page.

Step 2: `xsltproc`. This is the command-line program which takes your document and a PreTeXt stylesheet to together produce output. On Linux or a Mac you probably already have it installed as part of system software. On Windows it is not so simple.

In either case see the website for details abut verifying you have this, or how to install it.

2.3 Processing

At a command prompt in your terminal or console adjust the pathnames for the two files and execute:

```
xsltproc /path/to/mathbook/xsl/mathbook-html.xsl /path/to/quickstart.xml
```

In the current working directory you should now find the file `article-1.html` which you can view in a web browser. (You will want an internet connection since various parts of the page come from the network. Someday we will create output for the offline situation.) It will look very plain, but you should be able to read the sentence.

Now, try the following, again with adjusted paths:

```
xsltproc /path/to/mathbook/xsl/mathbook-latex.xsl /path/to/quickstart.xml
```

In the current working directory you should now find the file `article-1.tex` which you can process with `pdflatex` or `xelatex` at the command line as below. If you do not have LaTeX installed on your system, you could process this file within a variety of online services, and CoCalc would be an obvious choice.

```
pdflatex article-1.tex
```

In the current working directory you should now find the file `article-1.pdf` which you can view or print with standard PDF viewing software. You could even send it to a print-on-demand service to get nice hardback books, though I suspect sales will not be great.

That's it. You now know all the basics of authoring with PreTeXt, since you have produced two radically different output formats with identical content from the exact same structured input, via two different command lines. Everything you need to author a complete article or textbook, and produce it in many different formats, is just an extension or variation on what you just did. Let us look at a few simple extensions right away before being more methodical.

2.4 Extending the Minimal Example

We will not keep reproducing the entire example but instead suggest a series of modifications. After each edit, process the file again to make sure your syntax is correct (before you get too far along) and to see the changes.

The generic name of the resulting files is pretty bland, and it would be nice to have a title for our article. We will add an **attribute** to the `<article>` tag, specifically `@xml:id`, which is a very important part of PreTeXt and used frequently. For now, it will be used to generate the names of the output files. (The "at" symbol is a way of reminding you that it is an attribute, it is not part of what you author.)

So make the following modifications:

```
<article xml:id="quick">
    <title>My First Small Example</title>

    <p>This is a short sentence.</p>
```

Your outputs should now have a title, and more importantly, the filenames will be `quick.html` and `quick.tex`. Of course, you might like your outputs to have similar names to your input, but you see that this is not necessary.

Let us give our article a bit of structure. We will have an introduction and two sections with their own titles. So replace the one-sentence paragraph by the following, all following the article title and contained within the `<article>` tags. Remember, *do not* include any newlines (carriage returns, line feeds, hard wrap) in the longer lines. (We have to format things differently so you can see exactly what is happening.)

```
<introduction>
    <p>Let's get started.</p>
</introduction>

<section xml:id="section-short">
    <title>Beginnings</title>

    <p>This is a short sentence.</p>
</section>

<section xml:id="section-multiple-paragraph">
    <title>Endings</title>

    <p>This is a longer sentence that is followed by another sentence.
    Two sentences, and a second paragraph to follow.</p>

    <p>One more paragraph.</p>
</section>
```

The LaTeX/PDF output will be a bit odd looking since every paragraph is so short, but all the content should be there. Notice that the HTML output now has a table of contents to the left and active navigation buttons. Also the two sections are in their own files and the URLs have been constructed from the supplied values of the `@xml:id` attribute.

One last experiment—let us add some mathematics. We use XML tags, `<m>` for "inline" mathematics and `<me>` for a "math equation" which will be rendered with a bit of vertical separation and centered from left to right. We use LaTeX syntax for mathematics, which has been the standard for working mathematicians for decades. For electronic presentation, we rely on the excellent MathJax project which basically supports all the syntax of the amsmath package from the American Mathematical Society. Add the following sentence to any of the paragraphs of your article.

```
If the two sides of a right triangle have lengths <m>a</m>
and <m>b</m> and the hypotenuse has length <m>c</m>,
then the equation <me>a^2 + b^2 = c^2</me> will always hold.
```

So your final source file might look like the following. You now have many of the basic skills you would need to write an entire research article in mathematics, and should be in a position to learn the remainder easily and quickly.

```
<mathbook>
    <article xml:id="quick">
        <title>My First Small Example</title>

        <introduction>
            <p>Let's get started.</p>
        </introduction>

        <section xml:id="section-short">
            <title>Beginnings</title>

            <p>This is a short sentence.</p>
        </section>

        <section xml:id="section-multiple-paragraph">
            <title>Endings</title>

            <p>This is a longer sentence that is followed by another sentence.
            Two sentences, and a second paragraph to follow.</p>

            <p>One more paragraph.  If the two sides of a right triangle have
            lengths <m>a</m> and <m>b</m> and the hypotenuse has length <m>c</m>,
            then the equation <me>a^2 + b^2 = c^2</me> will always hold.</p>
        </section>
    </article>
</mathbook>
```

2.5 Where Next?

Next chapter has more detail.

Chapter 3

Overview of Features

This chapter is a high-level view of the important concepts, features and design decisions that go into the creation of PreTeXt. For careful exact descriptions of details, we will direct you to one of the many sections in the (*) Topics chapter. So this chapter should make you aware of what is possible and expand on the philosophy described earlier in Section 1.1, while also giving you examples of many basic constructions you can use to get started quickly.

3.1 Structure

A PreTeXt document is a nested sequence of **structural divisions**. For a book, these would go `<part>`, `<chapter>`, `<section>`, `<subsection>`, and `<subsubsection>`. Using `<part>` is optional, but a book must always use `<chapter>` (or else it is not a book!). No skipping over divisions. For example, you cannot divide a `<section>` directly into several `<subsubsection>`s without an intervening `<subsection>`.

An `<article>` starts divisions from `<section>`, though it may choose to have no divisions at all. `<paragraphs>` are exceptional. They lack a full set of features, but can be used to divide anything, in books or in articles, though they are always terminal since you cannot divide them further. You will have noticed that we prefer the generic term **division** (rather than "section") since a `<section>` is a very particular division.

A division may be unstructured, in which case you fill it with paragraphs and lists and figures and theorems and so on. But if you choose to structure a division it must look like an optional `<introduction>`, followed by multiple divisions of the next finer granularity, with an optional `<conclusion>`. Either version may have a single `<exercises>` division at the end. The structured version may have more than one `<exercises>`. Finally, there may be a single `<references>` within a division.

Every division tag can carry an `@xml:id` attribute, and it is a good practice to (a) provide one, (b) use a very short list of words describing the content, and (c) adopt a consistent pattern of your choosing. Do not use numbers, you may later regret it. These are optional, and with practice you will learn how best to use them. See Section 3.3 just below for more on this.

The `<exercises>` and `<references>` tags are special divisions, see Section 6.2 and Section 6.4.

This explanation is expanded and reiterated at Section 6.7 and is worth reading earlier rather than later.

3.2 Paragraphs

Once you have divisions, what do you put into them? Most likely, paragraphs. We use long, exact names for tags that are used infrequently, like `<subsubsection>`. But for frequently used elements, we use abbreviated tags, often identical to names used in HTML. So a paragraph is delimited by simply the `<p>` tag.

Lots of things can happen in paragraphs, some things can *only* happen in a paragraph, and some things are *banned* in paragraphs. Inside a paragraph, you can emphasize some text (`<em>`), you can quote some text (`<q>`), you can mark a phrase as being from another language (`<foreign>`), and much more. You can

use special characters like an ampersand (&, <ampersand/>) or an octothorpe, aka "hash tag" (#, <hash/>). You must put a list inside a paragraph, and all mathematics (Section 3.5) will occur inside a paragraph. You cannot put a <table> or a <figure> in a paragraph, and many other structured components are prohibited in paragraphs.

Paragraphs are also used as part of the structure of other parts of your document. For example, a <remark> could be composed of several <p>. As you get started with PreTeXt, remember that much of your actual writing will occur inside of a <p> and you will have a collection of tags you can use there to express your meaning to your readers.

So early in your writing project, familiarize yourself with the components of a paragraph detailed in Section 6.1.

3.3 Cross-References

Cross-references in a PreTeXt document are easy, powerful and flexible. So it is worth familiarizing yourselves with them early, here and then ahead in Section 6.6.

Any element that you place a @xml:id on can become the target of a cross-reference. This could be a division, a remark, a bibliographic entry, or a figure. So for example, suppose your source had <subsection xml:id="subsection-flowers"> and someplace else you wrote <xref ref="subsection-flowers" />. Then at the latter location you would get a reference to the Subsection that discusses flowers. In print this might just be the number for the subsection, but in various electronic output formats, these cross-references can be very powerful interactive ways to explore the content. And the mechanism is always the same, pair up an @xml:id on a target with a @ref on an <xref> cross-reference.

Since the value of an @xml:id is also used in a variety of ways, such as to construct some file names, some care should be taken in how you author them. We limit the possible characters to letters and numbers (a-z, A-Z, 0-9), with hyphens and underscores (-_) available as word-separators. Our advice is to stick to lowercase letters, though we are not yet aware of any problems with case-insensitivity. So in short, use **kebab-case** or **snake-case** for your @xml:id values.

For more, see Section 6.6 because cross-references have many features. But first, here are two features you do not want to miss. In the early stages of writing, you can author <xref provisional="subsection-flowers" /> to point to a subsection you are contemplating (but have not written yet) and you will get various polite reminders to get that straightened out eventually. Also the default behavior is to automatically provide the generic name of the target, so you will get something like "Subsection 4.3.2" without ever typing the "Subsection" part. If you move the target, the generic name will adjust if necessary, and if you switch to one of the supported languages, the generic name will switch language (see (((topic-on-languages))).

3.4 Titles

Divisions always need titles, you accomplish this with a <title> tag first thing. Almost everything that you can use in a paragraph can be used in a title, but a few constructions are banned, such as a displayed mathematical equation (for good reason). Try to avoid using footnotes in titles, even if we have tried to make them possible.

Lots of other structures admit titles. Experiment, or look at specific descriptions of the structure you are interested in. Titles are very integral to PreTeXt, much like cross-references. Titles migrate to the Table of Contents, get used in page headers for print output, can be used in lists (such as a List of Figures), and can be used as the text of a cross-reference, instead of a number. You might not be inclined to give a <remark> a title, but it would be good practice to do so. Your electronic outputs will be much more useful to your readers if you routinely title every structure that allows.

3.5 Mathematics

With experience, you may realize that PreTeXt utilizes three principal languages. One is the narrative of everyday sentences and paragraphs. Most of what you write in a paragraph, or a table cell, or a title, or a

caption, or an index heading, is in this language. Then there is the structural language, which is the majority of the elements in PreTeXt, such as `<chapter>`, `<theorem>`, or `<figure>`. Then finally, there is the language of mathematical symbols and notation.

A key design decision is that mathematical symbols, expressions and equations are authored using LaTeX syntax. More precisely, we support the symbols and constructions provided by MathJax, which quite closely follows the `amsmath` package maintained by the American Mathematical Society. Neither you nor I want to write MathML by hand!

For **inline** mathematics, use the short `<m>` tag within a `<p>` (or within a `<title>` or `<caption>`). For example, `<m>\alpha^2 + \beta^4</m>` will do what you expect, in print and in electronic outputs. To get a single equation, centered, with some vertical separation before and after, use the `<me>` tag ("math equation") in the same way within a `<p>`, but do not try using it within a `<title>`. For example, `<me>\rho = \alpha^2 + \beta^4</me>`. If you want your equation numbered, switch to the `<men>` tag (`n` = "numbered").

There is a way to incorporate your own (simple) custom LaTeX macros within mathematics (only). They will be effective in your print and electronic outputs, and can be employed in graphics languages like `tikz` and Asymptote. You can also author multi-line **display mathematics** using the `<md>` tag surrounding a sequence of `<mrow>` elements (or the `<mdn>` variant for numbered equations). We defer the details to Section 6.8.

3.6 Images

You can include an image via the `<image>` tag. You can use the `@source` attribute to provide a filename, likely prefixed by a relative path, that points at an image file. It is your responsibility to locate that file properly relative to your output, and that the file format is compatible. So, for example, suppose your source contained `<image source="images/butterflies.jpg" />`. Then you would want to have a directory named `images` below wherever you process your LaTeX output, or wherever you place your HTML output on a web server. The `@width` attribute can be used to control the size of the image. Widths are expressed as a percentage of the available width, such as `width="60%"`.

You may want to wrap your image in a `<figure>` to have it centered, and to have some vertical separation above and below. A `<figure>` must also have a `<caption>`, and the figure will be numbered.

If you want to place an anonymous image (no caption, no number), then you must wrap it in a `<sidebyside>`. Note also that the `<sidebyside>` tag provides some very flexible options for placing several images (Section 3.16) together, or combining figures with subcaptions.

If you wish to construct technical diagrams, with editable source, and perhaps including the use of LaTeX macros, PreTeXt provides support for graphics languages like PGF, TikZ, and Asymptote, in addition to using Sage code to describe a plot or image. In most cases output can be obtained as smoothly-scalable SVG images, in addition to other formats like PDF or PNG. Making all this happen is one of the more technical aspects of PreTeXt, so we save the discussion for one of the topics, (((topic-image-languages))).

3.7 Lists

Ordered lists (numbered), unordered lists (bullets) and description lists (defined terms) are all supported, and syntax generally follows HTML. Lists usually live within a paragraph (`<p>`), though there are limited exceptions. Their structure is given by the `<ol>`, `<ul>`, `<dl>` tags (respectively). These can specify a variety of options for the labels via attributes, as described in Section 6.9.

List items, for any of the three types, are delimited with the `<li>` tag. What is different from HTML is that the contents of a list item may be structured, with paragraphs (`<p>`) being the most likely. So to nest lists you begin a paragraph in a list item of the outer list, then begin the inner list within that paragraph. But a simple list item may be authored just like you were authoring within a paragraph, much like writing sentences elsewhere. For a description list, the list item must contain a `<title>`, which will become the text that is being described (the lable of the list item). Then the rest of the list item must be structured, say with paragraphs. A description list cannot be contained within another list.

Lists are more complicated than they appear, so be sure to read the details at Section 6.9 before you start designing rweally involved lists.

3.8 Exercises

Textbooks in many disciplines have exercises for the reader. In PreTeXt there are five places where you can set a question for the reader to pursue.

Divisional There is a special <exercises> division and an <exercise> placed there is then known as a **divisional exercise**. This division supports extra features designed for exercises, such as an <exercisegroup> for short exercises with common instructions. The <exercises> division can be used at any level. In other words, it can be a peer of any other division.

Inline Immediately within any division, you can interrupt the narrative with an **inline exercise**. It will be rendered similar to a <theorem> or other block, with a number, and a optional <title>.

Reading Question Another specialized division, <reading-questions>, can be used to house <exercise> designed to test or guide a reader's comprehension of the material in that division.

Worksheet The main component of a <worksheet> is an <exercise> (Section 3.9). Notably the exercises in a worksheet may be arranged with a <sidebyside> element (Section 3.16).

Project A <project> is a bit different than the prior situations, since this element is structured with <task>. But it has many common features, such as allowing optional <hint>, <answer>, <solution>.

If an <exercise> is simply a statement of the question, then it may be authored with paragraphs (<p>) and similar elements. If an <exercise> has hints, answers, or solutions, then it must be structured with a <statement>, followed by (possibly several) optional <hint>, <answer> and/or <solution>. Conceptually, an <answer> is a short final result, while a <solution> provides details about the route to the answer. Each of these four components is structured further, with paragraph-like elements, and the exercise itself may have a <title>. A title is strongly encouraged for inline exercises, and nearly-mandatory if you plan to have inline exercises rendered in knowls in a conversion to HTML.

You need (and want) to have the hints, answers and solutions grouped with the statement as you author, but there is a lot of flexibility on making these available at the location of the exercise, or in the back matter. See Section 6.10 for more.

An inline exercise typically gets a fully qualified unique number and is rendered similar to an <example> or a <remark>. A divisional exercise (including reading questions and worksheet exercises) only gets a sequential number, though this can be overridden with the @number attribute if you want to maintain stable numbering in response to edits. (Be careful, once you override the sequential numbering, you probably need to manually specify every subsequent number, so save overrides for when your project matures.)

Within a run of divisional exercises a subgroup can be delimited as an <exercisegroup>, which allows an <introduction> and/or <conclusion> to explain some commonality. An <exercisegroup> should be rendered in some way that makes it clear to the reader that they are a group.

3.9 Worksheets

Another division is a <worksheet>. It is similar to a <chapter>, <section>, and so on, but with some variations to support a worksheet or in-class activity. Here we recognize the primacy of printed output (perhaps to bring into a classroom), and the online version is a less-capable representation.

There is no limit to what you can place in a <worksheet> division: objectives, an introduction, theorems, figures, images, and so on. But the principal element is an <exercise>, which mostly behaves like an <exercise> in an <exercises> division, but with additional capabilities.

An <exercise> in a <worksheet> can have a specified width when included in a <sidebyside>, and in any case may have a specified additional blank working space of specified height. Page breaks can be specified, and the four margins on a page can be independently controlled. So if you want to create ancillary worksheets for your project, and you like to use *all* of the space on a printed page, then there is some layout control to support that.

Notice that all of this layout control is an exception to the philosophy of PreTeXt. So in particular, margins and working space do not appear in the HTML output. We do give a visual indication where a

page break in a <worksheet> is placed. An author might wish to collect all of the worksheets in a book, for printing as an "activity book", and so there are plans (2018-08-11) to support and automate that process. Details for authoring worksheets can be found in Section 6.3.

3.10 References

Like <exercises>, a <references> division may go anywhere a more typical division could go. This allows for things like a "Further Reading" list at the end of every chapter of a book. These are populated with <biblio> items that are individual bibliographic entries. Support is presently very minimal, but is planned to improve.

3.11 Figures and Tables

Figures and tables have outer structures, <figure> and <table> which can contain a <title> (for cross-references, such as in a list of figures) and a <caption>. The presence of a <caption> generates a number for the figure or table. These outer structures also generate some vertical separation between surrounding elements.

A <table> may only contain a <tabular>, which is the structure containing the rows and columns of a table. Details can be found in Section 6.12.

A <figure> is more liberal than a <table> and may contain a variety of items. Most probably, you will include an <image> (see Section 3.6).

3.12 Programs and Consoles

If you are writing about scientific or engineering topics, you may wish to include sample computer programs, or command-line sessions.

A <program> will be treated as verbatim text (see Section 3.14), subject to all the exceptions for exceptional characters. Indentation will be preserved, though an equal amount of leading whitespace will be stripped from every line, so as to keep the code shifted left as far as possible. So you can indent your code consistently along with your XML indentation. For this reason it is best to indent with spaces, and not tabs. A mix will almost surely end badly, and in some programming languages tabs are discouraged (e.g. Python).

The @language attribute may be used to get some degree of language-specific syntax highlighting. See Section 6.13 for details.

A <console> is a transcript of an interactive session in a terminal at a command-line. It is a sequence of the following elements, in this order, possibly repeated many times as a group: <prompt>, <input>, and <output>. The <output> is optional, and a <prompt> is only displayed when there is an immediately subsequent <input> (which could be empty). The content is treated as verbatim text (see Section 3.14), subject to all the exceptions for exceptional characters.

A <program> or <console> may be wrapped in a <listing>, which is analogous to a <figure> or <table>. A <listing> may contain a <title> for use in cross-references or lists. A <caption> will be displayed and will generate a number.

3.13 Special Characters

An advantage of XML syntax is that very few characters are reserved for the language's use, and thus very few characters need to be escaped. Of course, there is always the need to escape the escape character.

The escape character for XML is the ampersand, &. The other dangerous character is the left angle bracket, the "less than," <. Mostly to be symmetric, we also handle the right angle bracket, the "greater than," >, similarly. Single and double quotation marks are used to delimit attributes, so are part of the XML specification, but do not present difficulties in narrative text.

In normal writing, always use the empty elements <ampersand/>, <less/>, and <greater/>. Inside of

mathematics elements, or code for images written in LaTeX, always use the pre-defined macros, `\amp`, `\lt`, `\gt`. In verbatim text (such as programs) always use the XML entities `&`, `<`, `>`.

If you consistently follow the rules in the previous paragraph you will avoid a descent into escape-character hell and avoid a lot of head-scratching. In particular, you should have no need of the `<![CDATA[ ]]>` mechanism of XML, so just forget we even mentioned it.

Print and PDF output is generated via LaTeX, which has a good many special characters. So to preserve conversion to this format, you should consistently use provided empty elements for these characters. Here are the characters and their corresponding elements.

#	`<hash/>`
$	`<dollar/>`
%	`<percent/>`
^	`<circumflex/>`
&	`<ampersand/>`
_	`<underscore/>`
{	`<lbrace/>`
}	`<rbrace/>`
~	`<tilde/>`
\	`<backslash/>`

Table 3.13.1: LaTeX's reserved characters and their elements

There are some other empty elements, which are conveniences for certain characters, or sequences of characters, that are difficult or unusual in LaTeX and also somewhat obscure as Unicode characters. Two examples are the copyright symbol, ©, and constructions like the abbreviation "e.g." for *exempli gratia*. We will document these later.

3.14 Verbatim and Literal Text

Typesetting literal text, usually in a monospace font, can sometimes be tricky. For short bits of such text, as part of a sentence in a paragraph, or in a caption or title, use the `<c>` tag, which is short for "code." For much longer blocks of literal text, with line breaks that are to be preserved, use the `<cd>` element within a paragraph ("code display"). Outside a paragraph, most anywhere you could place a regular paragraph, use the `<pre>` tag, which is short for "pre-formatted".

For the content of a `<pre>` element, the indentation will be preserved, though an equal amount of leading whitespace will be stripped from every line, so as to keep the code shifted left as far as possible.

The behavior of these two tags is to preserve characters exactly. Certainly the ASCII character set will behave as expected, and Unicode characters will migrate successfully to output formats based on HTML. As mentioned in Section 3.13 the ampersand and left angle bracket will confuse the initial XML processing. So use the XML entities `&`, `<`, `>` to represent these characters to the XML processor, `xsltproc`. See Section 6.5 for further details.

3.15 Sage

Sage is an open source library of computational routines for symbolic, exact and numerical mathematics. It is designed to be a "viable free open source alternative to Magma, Maple, Mathematica, and Matlab." PreTeXt contains extensive support for including example Sage into your document.

A typical use of the `<sage>` tag is to include an `<input>` element, followed by an `<output>` element. The content of the `<input>` element may be presented statically in PDF output, or dynamically as a Sage Cell in an output format based on HTML. Of course, for output as a CoCalc worksheet, the Sage code is presented in the worksheet's native format.

The content of the `<output>` element is included in PDF output, but not in dynamic instances, since

it can be re-computed. Notably, there is a conversion which pairs input and output into a single file in the format used by Sage's `doctest` framework. So if expected output is provided, it becomes automatic to identify when Sage has diverged from your expectations, and you can adjust your examples accordingly.

The Sage Cell Server can also be configured to interpret different languages, because Sage by default contains everything needed to evaluate code in these languages. This is done by providing a `@language` attribute, where possible values are `sage`, `gap`, `gp`, `html`, `maxima`, `octave`, `python`, `r`, and `singular`. The default is `sage`.

Note that the dynamic formats (including the Sage Cell) may run Sage "interacts," so that is possible to embed interactive demonstrations into your dynamic output formats.

3.16 Side-by-Side Panels

A `<sidebyside>` is a useful organization of elements in a horizontal layout, and so begins to blur the line between content and presentation. While we default to organizing information in a vertical sequence, it is often desirable to organize smaller elements adjacent to each other horizontally. Specifically, images, tabular, figures, tables, paragraphs, and more, may all be combined and there is good control over vertical and horizontal alignment. Captioning, both overall and individually, is especially flexible. An `<sbsgroup>` ("side-by-side group") collects multiple `<sidebyside>` to stack vertically, which allows for displays in grids. See Section 6.14 for details.

3.17 Mathematical Results

Definitions, theorems, corollaries, etc. are supported by the tags: `<theorem>`, `<corollary>`, `<lemma>`, `<algorithm>`, `<proposition>`, `<claim>`, `<fact>`, `<definition>`, `<conjecture>`, `<axiom>`, and `<principle>`. Each may have a `<title>` (strongly encouraged), and then contains a `<statement>` which is a sequence of paragraphs and other elements. As appropriate, some of these elements (such as a `<lemma>`) may contain an optional `<proof>` (or several), while other elements may not have a `<proof>` (such as a `<conjecture>`).

A `<definition>` is a natural place to define notation as well (see Section 3.20), and to use the `<term>` tag to identify the terminology being defined.

In order to assist readers locating numbered items, these items are all numbered consecutively in a group that includes `<example>`s, `<remark>`s and inline `<exercise>`s.

3.18 Front Matter

In the beginning of your `<book>` or `<article>` you can have a `<frontmatter>` element that contains various items that would precede your first `<chapter>` or `<section>` (respectively). Possibilities include `<titlepage>`, `<colophon>`, `<biography>`, `<abstract>`, `<dedication>`, `<publication>`, `<foreword>`, and `<preface>`. Some of these may be duplicated (e.g. several prefaces for multiple editions), many of these items are restricted to books (e.g. a foreword), and some items are restricted to articles (e.g. an abstract). The DTD ((((dtd-info)))) will help you place them in the right order in your source. See Section 6.15 for details.

3.19 Back Matter

Similar to front matter, there is material you might wish to include after your book's final `<chapter>` or your article's final `<section>`. Possibilities to place in a `<backmatter>` include `<appendix>`, `<references>`, `<index-part>`, and `<colophon>`. There are empty tags you can place into an appendix to generate lists of notation, or lists of particular elements of your choice, such as a list of figures. A similar empty element actually generates the index, `<index-list/>`. See Section 6.15 for details on the back matter generally, and Section 6.18 for more on automatic lists.

3.20 Index and Notation Entries

Construction of an index and a list of notation is accomplished by placing information into your text in the appropriate places in the right way.

The <idx> tags denote an index entry. These should be placed within the element that they describe. By this we mean that an <idx> element can be placed within a <theorem> to refer to just that theorem, or it might be placed within a <subsection> to refer to that subsection. In this way, electronic versions of your work can have an index that is more informative than a traditional index that uses just page numbers. Note that the text contained within the <idx> tags does not actually appear in the article—it only serves to mark the location the index entry points to. You can have several levels of headings by structuring your <idx> element with up to three <h> tags. Alternatively, after the first use of an <h> element, you can use <see> or <seealso> to denote a "see" or "see also" in the index entry, respectively.

A similar device is used to create a list of notation for a technical (mathematical) work. Place a <notation> element as close as possible to the place where notation is first introduced. If you use the <definition> tag for your definitions, then this is a very natural place to also introduce notation. Inside of <notation> use the <usage> tag to include a short example of the notation in use. This will be treated as mathematics, so imagine that it will be wrapped in an <m> tag and use LaTeX syntax. The <description> tag should contain a very short description in words of what the notation is for. So "center of a group" would be a good description to accompany the usage "$Z(G)$."

3.21 WeBWorK Exercises

It is possible to author WeBWorK automated homework problems directly within your source. A static version will be rendered in your LaTeX/PDF/print output, and a "live" version will be rendered into HTML output (though a student is not able to authenticate against a course). However, you can also extract *all* of the WeBWorK questions from a textbook into a single archive suitable for uploading into a traditional WeBWorK server. This is a big topic, so see the dedicated Chapter 10 for details.

3.22 URLs and External References

The <url> tag always requires an @href attribute. Usually this will be some external web page, or other resource. If the <url> is empty, then the value of the @href attribute will be the link text, typically in a monospace font. You can also provide your own content, using elements much like any other piece of text that would occur in a paragraph. Of course, there is no need to anticipate any problems with LaTeX output and special characters like a tilde. Use the character itself in the @href, and use the <tilde> element in the content (or wrap the content in the <c> element and use the literal character).

This element may also be used to link to external data files. See Section 6.19 for details.

3.23 Video

Videos, either author-hosted, via a URL, or hosted on YouTube, may be embedded in a PreTeXt document that is converted to HTML, and may be optionally "popped-out" to view on another page. For a YouTube video, it is simplicity itself, as an author need only supply the identification string, and all the details of the embedding are handled by PreTeXt. See Section 6.20 for details.

3.24 Scientific Units

If you are writing about science or engineering, or even if you are not, there is extensive support for scientific units. So, for example, you could author a force in metric units as

```
<quantity>
    <mag>20.7</mag>
```

```
    <unit prefix="kilo" base="gram" />
    <unit base="meter" />
    <per  base="second" exp="2" />
</quantity>
```

This would be rendered as 20.7 $\frac{\mathrm{kg\,m}}{\mathrm{s}^2}$. More in Section 6.22.

3.25 Accessibility

The Web Accessibility Initiative at W3C (`www.w3.org`) says:

> The Web is fundamentally designed to work for all people, whatever their hardware, software, language, culture, location, or physical or mental ability. When the Web meets this goal, it is accessible to people with a diverse range of hearing, movement, sight, and cognitive ability.
>
> Thus the impact of disability is radically changed on the Web because the Web removes barriers to communication and interaction that many people face in the physical world. However, when websites, web technologies, or web tools are badly designed, they can create barriers that exclude people from using the Web.

Since we are interested in helping authors produce documents with open licenses, and we concentrate on employing open standards for the HTML output we create, we are ideally positioned to help you easily create highly-accessible documents. There are many technical features which happen automatically, and there are some features which we make available for your use as an author and which only an author can provide. See Section 6.26 for full details and ways you can make the HTML version of your document more accessible, and more useful for a wider audience.

Chapter 4

Processing, Tools and Workflow

This chapter explains in full detail how to combine your source file with an XSL stylesheet to produce output. It expands on the simple example in Chapter 2 and should also be read in conjunction with the chapter on the `mbx` script (Chapter 9).

4.1 Basic Processing

The executable program `xsltproc` implements Version 1.0 of the **eXtensible Stylesheet Language (XSL)**. This is a declarative language that walks the hierarchical tree of an XML source file, and for each element describes some output to produce before, and after, recursively processing the contained elements. (That is a simplified description.)

`xsltproc` is typically installed by default on Linux systems and as part of Mac OS. See the PreTeXt website for details for Windows systems. The most basic operation is to provide `xsltproc` with an XSL stylesheet from the PreTeXt distribution and an XML document of your creation that is valid PreTeXt. This is done at the command-line, inside of a terminal or shell. Describing command-line operations, along with file and directory management, is beyond the scope of this guide, so consult another resource if this is unfamiliar. So here is a hypothetical simple example:

```
rob@lava:~/mathbook$ xsltproc xsl/mathbook-html.xsl ~/books/aota/animals.xml
```

By default, `xsltproc` writes output to `stdout` (the screen), which you could redirect to a file, or you could use the `-o` switch to send the output to a named file. However, PreTeXt automatically writes to a file whose name is derived from the `@xml:id` attribute of the top-level `<book>` or `<article>` tag. If no such attribute is given the filename will be derived from `book-1` or `article-1`. All output is produced in whatever the current default directory is, so you will likely want to set this beforehand.

The `xsl` subdirectory of the PreTeXt distribution contains a variety of XSL stylesheets, which I will also refer to as **converters** or **conversions**. The ones that you will use as an author all have filenames of the form `xsl/mathbook-XXX.xsl`, where `XXX` is some indication of the output produced. Conversions to LaTeX or HTML output are the two most mature converters.

Note that authors are not responsible for creating XSL stylesheets. Stock conversions are part of the PreTeXt distribution, and anybody is welcome to assume a source document is valid PreTeXt and create new conversions to process it to existing, or as yet unimagined, formats.

4.2 Modular Source Files

For a large project, such as a book, you will likely want to split up your source into logical units, such as chapters and sections. `xsltproc` supports an include mechanism that makes this possible. Let us suppose that your `book` on animals has a `chapter` on mammals with a `section` on monkeys. Then you need to do the following:

1. For the file containing the <chapter> tag for the chapter on mammals, place the attribute

```
xmlns:xi="http://www.w3.org/2001/XInclude"
```

   on the outermost tag in the file.

2. Within the <chapter> element for the chapter on mammals, add the line <xi:include href="monkeys.xml" /> to "pull in" the section on monkeys at that location. The @href attribute can point to a file in a subdirectory, but will be interpreted relative to the location of the file containing the mammal chapter element.

3. Add the switch -xinclude to your invocation of xsltproc, just after xsltproc, but before the filenames for the stylesheet and the master source file. Note that for some versions of xsltproc it might be necessary to use two dashes for the switch, --xinclude.

So now a typical invocation (using one dash) might look like

```
xsltproc -xinclude xsl/mathbook-html.xsl ~/books/aota/animals.xml
```

Note that when you invoke xsltproc the default directory can be far away from your source, and the processor will locate all the component files of your project through the relative file locations in the @href attribute. Several comments are in order.

- Begin small and start a project *without* using modular files. Modularizing seems to add a layer of complexity that sometimes obscures other beginner's errors. So get comfortable with a single source file before branching out.
- I am forever forgetting the -xinclude switch. Empty output, or cryptic error messages, are your first clue to this simple, but common, mistake.
- The XML specification requires that a source file only contain a single outermost element. So for example, two <chapter> elements cannot go into the same file as simultaneous outermost elements.
- Any file that uses an <xi:include> element will need the xml:ns delaration on the outermost element. So in our animal book example, the "master" file, which presumably includes several chapter files, would need this declaration on the <mathbook> element.
- In practice, there is not a lot to be gained by creating a subdirectory structure mirroring your modularization—all your source files can go into one big directory and the XML hierarchy will take care of the organization. I do sometimes like to name my files accordingly, so for example chapter-mammals.xml and section-monkeys.xml.

The sample book in examples/sample-book amply demonstrates different ways to modularize parts of a project (but in no way should be taken as best practice in this regard). This guide, in doc/author-guide is a simple example of modular source files, and might be a good template to follow for your book. See Section 6.25 for some of the finer points of this topic.

4.3 Verifying your Source

A **schema**, in our case a RELAX-NG schema, is a formal specification of an XML vocabulary (the allowed tags and attributes), and how they relate to each other. So, for example, the restrictions that say you cannot nest a <book> inside of a <chapter>, nor can you nest a <subsection> in a <chapter> without an intervening <section>, are expressed and enforced by the schema. One of the beauties of the schema is that it is written using a very specific syntax and then there are tools that use a schema as input. In particular, a PreTeXt source file that conforms to the PreTeXt schema is said to be **valid**. You should strive to always, always, always have valid source files, and therefore you want to regularly verify that this is the case.

You can find the PreTeXt schema in the schema directory. The version we author and maintain is pretext.xml, which is used to create pretext.rnc, which uses the compact syntax of the RELAX-NG

specification. By providing the schema and your source to a program called a **validator** you can check if your source is valid, and if not, why. See Chapter 5 for the details on doing this.

If you author source that is valid PreTeXt, then a conversion of your source to another format should succeed. And maybe in the future, somebody will create a new conversion to a new output format, and your source should still produce faithful output, with no extra effort from you. Think of the schema as a contract between authors of source files and developers of converters. This is different than performing a conversion and getting good-looking output–that can just be a happy accident and your source may not succeed with some other conversion.

We cannot stress enough the importance of setting up and performing regular validation and preventing many consistent errors of the same type. You will learn what elements are allowed where, and which are not, from the messages produced by validation errors. And when a conversion fails, or produces spectacularly incorrect output, validating your source should be your first reaction. Always.

The other beauty of a schema is that you can supply it to a text editor (Section 1.2) and then you will get context-sensitive help that greatly assists you in using only the tags and attributes that are allowed in a given location of your source. XML Copy Editor is the one editor like this we have tried, but we do not have extensive experience.

We have devoted an entire chapter (Chapter 5) to amplifying this introduction and providing more details, such as where to find details on installing a validator.

4.4 Customizations, String Parameters

There are some aspects of your output that are entirely divorced from the actual content, and are presumably all about how that content is presented. Two good examples are the size of the font used in LaTeX/PDF/print output, and the granularity of web pages in HTML output (by this we mean, is each web page a whole chapter, a whole section, a whole subsection?). Producing output with varying values of these parameters does absolutely nothing to change your content in any way, and so should not be a part of your source. Thus we provide values for these parameters on the command-line *at processing time*. They have become known as **stringparam** for a soon-to-be obvious reason.

Suppose you want to make a large-font version of your textbook for a student who has limited vision. Look inside the top of `xsl/mathbook-latex.xsl` and find the `latex.font.size` parameter. The preceding comments in this file suggest `20pt` is the maximum supported. So you would use a command-line like the following (possibly with `--xinclude`, etc.).

```
$ xsltproc --stringparam latex.font.size "20pt"
/path/to/xsl/mathbook-latex.xsl ~/books/aota/animals.xml
```

You can use as many stringparam as you like on the command-line (or in your scripts). The quotation marks are not strictly needed in this example, but if the value of the parameter has spaces, slashes, etc., then you need to quote-protect the string from the command-line processor, and either single or double quotes will work (and protect the other kind).

These parameters are documented in the XSL files themselves, principally `-common`, `-latex` and `-html`, and occur near the top. They assume sensible defaults for beginners, and error-checking is careful and robust. They will be easier to locate and use when we have the time to document them more carefully here in the Author's Guide.

One caveat for using these is that experience has taught us that some of the parameters we created early on really do affect your content. We will change some of these, but always provide a smooth upgrade path through deprecations, with little or no disruption to your workflow.

4.5 Customizations, Thin XSL Stylesheets

Stringparams (Section 4.4) are an easy way to effect global changes in the presentation of your writing. But putting ten of them on every command-line gets old and cumbersome fast.

You may also wish to customize your output in some stylistic way. This might be especially true for LaTeX/PDF/print output. For example, you might wish to have every chapter heading of your book in a nice

shade of light blue, with the title flush right to the margin, countered by a thick solid rule extending all the way right, to the edge of the paper. Notice that this does not affect your content, it is strictly presentation.

We have done several things to encourage such customizations. We have tried to put as much stylistic information as possible in the LaTeX preamble and keep as much as possible out of the body. (There is always room for improvement on this score, please be in touch if you have a need.) For small adjustments the `latex.preamble.early` and `latex.preamble.late` stringparam are possible vehicles, though all the LaTeX code to make light blue, flush-right rules is going to be messy on the command-line.

Instead, you can make a small XSL file, to use as input to `xsltproc`. The first thing it should do is import the stock PreTeXt file for the type of output you want to create. You can use an absolute path to the PreTeXt distribution (which will not be very portable), or utilize the `mathbook/user` directory and a relative path from there. The easiest thing to put in this file is elements like

```
<xsl:param name="latex.font.size" select="'20pt'" />
```

which is functionally equivalent to our example in Section 4.4. Values given on the command-line supersede those given in an XSL file this way.

You can augment the LaTeX preamble with as much LaTeX code as you like in the following way.

```
<xsl:param name="latex.preamble.late">
    <xsl:text>% Proof environment with heading in small caps&#xa;</xsl:text>
    <xsl:text>\expandafter\let\expandafter\oldp\csname\string\proof\endcsname&#xa;</xsl:text>
    <xsl:text>\let\oldep\endproof&#xa;</xsl:text>
    <xsl:text>\renewenvironment{proof}[1][\proofname]{\oldp[\scshape #1]}{\oldep}&#xa;</xsl:text>
</xsl:param>
```

There are a variety of things you can do generally, by overriding the imported XSL templates to change behavior, but such modifications are beyond the scope of this guide.

4.6 Images and the `mbx` Script

We believe it is important to preserve a record of how diagrams and other graphics are produced. This can be easy when a graphics language is employed to describe the graphical elements, rather than creating a bit-mapped image with some other interface. So we have `<asymptote>`, `<latex-image-code>`, and `<sageplot>` for elements holding code to produce diagrams or images.

The upside to this is that small edits to the code can easily accomplish minor changes or corrections necessary for the images. The LaTeX macros provided by an author can be used in the text *and* in a diagram, leading to greater consistency between the two. Finally, starting from source, we can do the best possible job of producing image formats that are compatible with the document output formats and which scale smoothly in PDFs and in web browsers.

The downside to this is that XSL is not a general purpose programming language, and so in particular, cannot call "helper" programs such as `asy`, `pdflatex`, and `sage`. The general strategy is to use XSL to identify and isolate the parts of a document that lie in the elements designed for graphics languages. A Python script, the `mbx` script, employs these XSL stylesheets and then feeds each image file to the appropriate helper program.

This script has a variety of options, so we document it fully in Chapter 9.

4.7 Keeping Your Source Up-to-Date

Once in a while it becomes necessary to adjust how the PreTeXt vocabulary is arranged, which involves adding or removing elements or attributes, or changing their behavior. When elements or attributes are removed, or their relationships with other elements change, we say that certain items or behaviors are **deprecated**. Fortunately, we can often automate the changes.

When there is a deprecation, a warning is added so that any conversion will report the presence of the old use in the console. Sometimes we can preserve the old behavior, so there is no rush to make changes to

your source. Sometimes a change needs to be more urgent. And frequently old behaviors do not get updates or bug-fixes. Our warnings provide advice and information about what you need to do. There are also announcements on public discussion groups, clearly marked as deprecations. Also, the schema will change as part of any deprecation, so the old elements or old use will be reported.

The rest of this section describes a tool you can use to automate the process of adjusting your source when there are deprecations. Generally, there is an XSLT stylesheet which will convert your XML source to another XML source file, fixing many of the deprecations automatically. However, it is the nature of XML processing that your source file will undergo some cosmetic changes. For example, the order of attributes is never relevant, so an XML-to-XML conversion is free to re-order the attributes of an element, perhaps different from how you like to author them.

So you have two choices:

- Process your source with any of the provided conversions and edit by hand until the warnings all disappear.
- Run the deprecation-fixing conversion and accept the changes in XML formatting. (Read on for more specifics about these changes.)

You perform this conversion using `xsl/utilities/fix-deprecations.xsl` on an XML source file in the usual way. By default, output appears on the console, so you will want to specify an output file, for example with the `-o` flag of `xsltproc`. You will discover a safety measure that requires you to also use a parameter, which you can pass in to `xsltproc` with the `-stringparam` command-line argument.

One choice of the parameter will result in just "copying" your source file and making all the cosmetic source format changes (we refer to this here as **normalization**). This might be a useful thing to do first, all by itself, either as a first step, or an exploratory experiment. The other value of the parameter will actually make changes, and report some information about progress.

Here are some notes:

- Be sure to experiment on copies of your source in a scratch directory. Send your output to another directory. When finished, use a `diff` tool to inspect the actual changes made. You can record your eventual changes using revision-control (Appendix G).
- Do not enable `xinclude` processing or else your several files will all be merged into one as output and any modularity of your source will be lost.
- Every single bit of indentation and whitespace in your source will be preserved, except perhaps for some blank lines near the top of your source files, and limited exceptions noted below.
- Attributes will likely be re-ordered, with normalized spacing between them.
- Empty elements will have any spaces removed from the end of the tag.
- Elements with no content may be written with a single empty tag.
- CDATA sections will be converted to text acceptable to the XML parser. In other words, the CDATA wrapper will be removed and dangerous characters (&, <, >) will be replaced by equivalent entities (such as `&`). If you have many matrices expressed in LaTeX and wrapped in a CDATA, this might be a big change.
- The output files will be labeled as having `UTF-8` encoding.
- It could be necessary to run this conversion more than once if deprecations build on one another. In other words, we do not update specific conversions, but rely on regular use to keep source up-to-date.
- It should be safe to run this conversion repeatedly, even after new deprecations are added. In fact, it is encouraged.
- The PreTeXt source file `examples/sample-errors-and-warnings.xml` is intentionally full of lots of bad stuff. You can experiment with it, should you want to see interesting things happen. We have already performed the normalization step, so you can concentrate on substantive changes.

To process a directory with multiple source files, I would proceed as follows. First make three temporary directories, `/tmp/original`, `/tmp/normal`, `/tmp/clean`, and copy my source files into `/tmp/original`. Then, using a BASH shell, and inputting the command all on one long line,

```
rob@lava:/tmp/original$ for f in *.xml; do xsltproc -o ../normal/$f -stringparam fix normalize
/home/rob/mathbook/xsl/utilities/fix-deprecations.xsl $f; done
```

This will loop over every XML file in the current working directory, `/tmp/original`, running the normalization conversion on each file, with the output files using the same filename, but now being placed in the `/tmp/normal` directory. If you change to the `/tmp` directory, then you can compare the results. I like to use the `diff` utility provided by `git`.

```
rob@lava:/tmp$ git diff original normal
```

Or, try this for a view that might be more informative.

```
rob@lava:/tmp$ git diff --word-diff original normal
```

You may only do the above once, on your first use of this conversion stylesheet. You will see how your style of authoring XML will undergo some minor changes. We can repeat the above to actual make the changes necessary due to PreTeXt deprecations. Make `/tmp/normal` the working directory.

```
rob@lava:/tmp/normal$ for f in *.xml; do xsltproc -o ../clean/$f -stringparam fix all
/home/rob/mathbook/xsl/utilities/fix-deprecations.xsl $f; done
```

And as above, you can now compare the `normal` and `clean` directories to see actual changes. If you are satisfied with the changes, you can copy the files in the `clean` directory back onto your source files. If you are using revision-control (you are, aren't you?) then you can make a commit that holds these changes (Appendix G). Or maybe even make two commits, one from the normalization step, and a second with the substantive changes.

4.8 File Management

PreTeXt, at its core, is the formal specification of the XML vocabulary, as expressed in the DTD (Section 4.3). We have provided converters to process source files into useful output. However, we have not yet built a point-and-click application for the production of a book. So you need to take some responsibility in a large project for managing your files, both input and output. We have tried to provide flexible tools to make an author's job easier. The following is advice and practices we have successfully employed in several book projects.

Source. I am fond of describing my own books with an initialism formed from the title. So *A First Course in Linear Algebra* becomes FCLA, and in file and directory names becomes `fcla`. So I have a top-level directory `books` and then `books/fcla`, but this directory is not the book itself, this is all the extra stuff that goes along with writing a book, much of it in `books/fcla/local`. The actual book, the part everybody sees with an open license, lives in `books/fcla/fcla`. This subdirectory has files like `COPYING`, which is a free software standard for license information, and `README.md` which is a file in the simplistic Markdown format that is picked up automatically by GitHub and displayed nicely at the book's repository's main page. Subdirectories include `src` for the actual XML files, `xsl` for any customizing XSL (Section 4.5), and `script` for shell scripts used to process the book (see below).

I do not use any additional directory structure below `src` to manage modular files for a book, since the XML and the `--xinclude` mechanism manage that just fine. I see little benefit to extra subdirectories for organization and some resulting inconvenience. I do typically have a single subdirectory `src/images` for raster images and other graphics files.

I believe it is critically important to put your project under revision control, and if licensed openly, in a public GitHub repository. So the `books/fcla/fcla` directory and all of its contents and subdirectories is tracked as a `git` repository and hosted on GitHub. Because this directory is *source* I try very hard to *never* have any temporary files in these directories since I do not want to accidentally incorporate them into the `git` repository. As a general rule-of-thumb, only original material goes in this directory and anything that

can be re-created belongs outside.

A tutorial on `git` would be way outside the scope of this guide, but Beezer and Farmer *have* written *Git For Authors*, so perhaps look for that.

Image Files. Some images are raster images (e.g. photographs) that are not easily changed, and perhaps unlikely to be changed. Other images will come from source-level languages via the `mbx` script. For your convenience, this script has a command-line option that allows you to direct output (graphics files) to a directory of your choice.

In the early stages of writing a book, I put image files produced from source code in a directory outside of what is tracked by `git`. It is only when a project is very mature that I begin to include completed graphics files into the `src/images` directory for tracking by `git`.

Build Scripts. When you have a mature book project, the various files, processing options, and a desire for multiple outputs can all get a bit confusing. Writing simple scripts is a good idea and the investment of time doing this early in a project will pay off through the course of further writing and editing. The particular setup you employ is less important.

I have fallen into the habit of using the `make` program. It allows me to define common variables upfront (such as paths to the PreTeXt distribution and the main directory for the project it applies to). Then I can easily make "targets" for different outputs. So, for example I typically go `make pdf` or `make html` to produce output, and have simple companion targets so that I can go `make viewpdf` or `make viewhtml`. Other targets do things like checking my source against the DTD (Section 4.3). I have split out the variable definitions in a way that a collaborator can join the project and simply edit the file of definitions just once to reflect their setup, and still participate in future upgrades to the script by pulling from GitHub and not overwrite their local information.

My use of `make` is a bit of an abuse, since it is really designed for large software projects, with the aim of reducing duplicative compilations and that is not at all the purpose. You could likely have exactly the same effect with a shell script and a case (or switch) statement.

My general strategy is to assemble all the necessary files into a temporary directory (under `/tmp` in Linux) by copying them out of their permanent home, copy customizing XSL into the right place (typically `mathbook/user`), run the `mbx` script as necessary and direct the results to the right place, and finally copy results out of the temporary directory if they are meant to be permanent. Interesting, an exception to staging all these files is the source of the book itself which is only read for each conversion and then not needed for the output. So you can just point directly to a master file and the `xinclude` mechanism locates any other necessary source files.

A good example of this general strategy is the use and placement of image files for HTML output. It is your responsibility to place images into the location your resulting HTML files expect to locate them. By default, this is a subdirectory of the directory holding the HTML files, named `images`. You will want to copy images, such as photographs, out of your main source directory (`src/images`?). But you may be actively modifying source code for diagrams, and you want to re-run the `mbx` script for each run, and make sure the output of the script is directed to the correct subdirectory for the HTML output. Running the `mbx` script frequently can get tiresome, so maybe you have a makefile target `make diagrams` that updates a permanent directory, outside of your tracked files in the repository, and you copy those files into the correct subdirectory for the output. That way, you can update images only when you are actively editing them, or when you are producing a draft that you want to be as up-to-date as possible. As a project matures, you can add images into the directory tracked by `git` so they are available to others without getting involved with the `mbx` script.

We did not say it would be easy, but we feel much of this sort of project management is outside the scope of the PreTeXt project itself, while in its initial stages, and existing tools to manage the complexity are available and documented. (We *have* been encouraged to create sample scripts, which we may do.) Just remember the strategy: stage necessary components in a temporary directory, build output in that directory, copy out desired semi-permanent results, and limit additions to the source directory to that which is original, or mature and time-consuming to reproduce.

4.9 Testing HTML Output Locally

Certain complicated parts of HTML output will not always function when you look at PreTeXt output by just opening files in your web browser. These include knowls, Sage cells, and YouTube videos. This is a consequence of security policies and so will vary from browser to browser. A solution is to run a web server on your own machine, which is much easier than it sounds. The one prerequisite is that you have Python installed, as is normally the case on any Linux or Mac computer. On Windows, you may need to install Python yourself, but you may eventually need it for the `mbx` script anyway (Chapter 9). The following has been tested on Ubuntu Linux 16.10 and MacOS 10.12 (2017-08-06).

Python 2 and Python 3 are different in some regards, and that is the case here. Your system may have commands `python2`, `python3`, and/or plain `python`. Experiment with variants of the following two commands. First, at a command-line, set your working directory to be the location of a directory containing the HTML files you want to test.

```
python3 -m http.server

python2 -m SimpleHTTPServer
```

Use `--help` if you want to see the (limited) options for configuration. Then go to the address bar of your browser and use

```
http://0.0.0.0:8000/index.html
```

to actually view the files from a web server. Running the Python command should tell you which address to use (the `0.0.0.0`) and your project might not have an `index.html` file, so adjust accordingly.

Official documentation: Python 3 and Python 2.

You may also be able to configure your `hosts` file so that this webserver looks like it lives somewhere else. Clues at `bowerwebsolutions.com`.

4.10 Doctesting Sage Code

Adding computer code to your textbook is a tricky proposition. You can propose that it is merely an illustration, and not meant to have all the necessary details, or you can make it exact, correct and executable, and then risk inevitable changes to render your code obsolete. At least you have the option of editing and reposting online versions quickly and easily.

One of our main motivations for this project was mixing in code from the powerful, open source, mathematical software project, Sage (Section 3.15). When you add example Sage code to illustrate mathematical ideas, you are then encouraged to also include expected output in the `<output>` element. Here comes one of the powerful advantages of XML source and XSL processing.

The `mathbook/xsl/mathbook-sage-doctest.xsl` stylesheet, used in the usual way, will create one (or several, depending on the `chunk.level` stringparam) file(s), in *exactly* the format Sage expects for automated testing. So all your words are gone, and all your Sage input and output is packaged so Sage can run all the `<input>` and compare the results to the expected `<output>`.

We have many years' experience testing hundreds of non-trivial Sage examples from textbooks, for linear algebra and abstract algebra. Roughly every six months, we discover ten to twenty examples that fail. Frequently the failures are trivial (usually output gets re-ordered), but some are significant changes in behavior that leads us to re-word surrounding guidance in the text, and in a few cases the failures have exposed bugs introduced into Sage. It has been relatively easy to do this maintenance on a regular basis, and if it had not been done, the accumulated errors would be enough to greatly degrade confidence in the accuracy of the examples.

Exact details for this process can be found in Section 6.24. Note that Sage is really just a huge Python library, so it might be possible to test pure Python code with this facility, but we have not tested this at all. Similar support for other languages can be considered if requested for use in a serious project.

4.11 Author Tools

The general stringparam `author-tools` may be set to `yes` to activate special handling of the `<todo>` element, the `@provisional` attribute of an `<xref>` element, and notation and index entries. The LaTeX-specific stringparam `latex.draft` set to `yes` will automatically activate the previous features, in addition to a few others appropriate to the printed page. We have an XSL stylesheet that dumps every `<todo>` and every `@provisional` attribute to the screen as a simply-formatted text report, but it is not public now.

The intent here is to make a rough draft, for an author or collaborator only, reporting as much as possible that is incomplete, pending or hidden in the usual output. But none of this is carefully supported. Requests for improvements, and overall comments, from working authors are invited. They will help to make this feature more useful for everybody.

4.12 Building Output in CoCalc

CoCalc `cocalc.com` has *all* the tools you need to author with MathBook XML. You will need an upgrade from a subscription to allow Internet connectivity, but at a minimum a colleague with a paid plan can spare you one, they are plentiful and meant to be shared.

- Text editor: reasonably good, partially XML syntax-aware.
- `git`: installed (so clone PreTeXt).
- LaTeX: installed with many additional packages.
- Python: installed, necessary for `mbx` script.
- PDF viewer: handed off to your browser.
- HTML viewer: convert to the "raw" URL and you can preview.
- HTML server: nope. Zip up output and host elsewhere.

Chapter 5

PreTeXt Vocabulary Specification

This Author's Guide, along with the sample article and sample book distributed with the PreTeXt source, provide a wealth of examples of how to author in PreTeXt. However, at some point, you will undoubtedly encounter a situation where some of your text fails to appear in your output or `xsltproc` produces an error. Those are good moments to start investigating the formal specifications of the PreTeXt vocabulary, as most likely you tried to use something in a way incompatible with those specifications. This chapter will help you understand, and work with, the formal specification of PreTeXt.

5.1 RELAX-NG Schema

A **schema** is a set of patterns which describe how the elements of a language may be combined. The PreTeXt vocabulary is described by a RELAX-NG schema, which is included in the PreTeXt distribution. (RELAX-NG stands for "REgular LAnguage for XML Next Generation" and is pronounced "relaxing".) In general terms, the schema tells you which elements are available, which attributes they may carry, and which other elements they may contain. You can then infer where you can place an element. The schema also indicates if an element is required or optional, if it may be repeated, or if it needs to appear in a prescribed order with other elements, and may limit attribute values.

Besides providing a concise formal description of the PreTeXt vocabulary, your XML source and the RELAX-NG schema can be provided to tools which will automatically **validate** your source against the formal definition. The best validators will provide accurate and helpful messages about errors in your source. Further, some editing tools will use a schema to provide context-sensitive assistance on available elements and attributes as you write, sparing you typographical errors, misplaced elements, and the need to frequently context-switch in order to consult reference material.

The schema does not tell you anything about how an element or attribute will behave. But hopefully there is not much ambiguity about the behavior of the content of a `<title>` element nested within a `<chapter>` element. You would not be surprised to see that content duplicated in the Table of Contents of a book. The purpose of this guide, and other documentation, is to help you understand what to expect. It is better to think of the schema as a contract between you and the developers of conversion tools. If your source conforms to the schema, then a conversion tool will produce reasonable output that conveys the structure and meaning of your writing. Twenty years from now, when GEFF is the dominant document format, a conversion of your source will preserve your meaning, while also taking advantage of the amazing features of GEFF. (GEFF stands for "Great Electronic Format of the Future".)

In summary, the RELAX-NG schema

- is the formal specification of the PreTeXt vocabulary,
- is a key input to validation,
- can be incredibly helpful in the editing process, and

- provides guidance to implementors of conversions.

As such, we are very deliberate about changes, and hope over time to make changes only very rarely.

5.2 Schematron Rules

The RELAX-NG schema is very good at specifying "parent-child" relationships, in other words, which elements can nest directly under/within other elements. But we have situations where the possible elements depend on grandparents, great-grandparents, or older ancestors. An example is the `<var>` element, which is only useful if it is contained *somewhere* within a `<webwork>` element. You can describe these situations with RELAX-NG, but it becomes cumbersome and redundant. So our stategy is to allow some prohibited situations in the RELAX-NG schema, and use Schematron rules to identify the prohibited situations. Continuing our WeBWorK example, the RELAX-NG schema makes it appear that `<var>` can be used many places, but a Schematron rule will provide a helpful message indicating you have used it outside the context of a WeBWorK problem.

> Schematron is a feather duster to reach the corners that other schema languages cannot reach.
>
> —Rick Jelliffe

5.3 Versions of the Schema

The schema is born within a PreTeXt document, `schema/pretext.xml`, where surrounding text provides documentation and guidance on implementation. The literate programming support in PreTeXt (see (((literate programming documentation)))) is used to produce a file, `schema/pretext.rnc`, which is a RELAX-NG specification in compact syntax. As the literate programming support improves from 2017-07-21, HTML and PDF versions will be made available on the PreTeXt website as documentation. We provide some guidance below on reading the compact syntax.

The compact syntax is a faithful representation of the more verbose XML syntax. And vice-versa, so it is possible to translate back-and-forth between the two syntaxes. In practice, we convert the compact version to the XML version, producing a file `schema/pretext.rng`. Some tools require this latter (100% equivalent) version. We perform this conversion with `trang`, an open source program written by James Clark, one of the authors of RELAX-NG. (`trang` stands for "TRAnslator for relax NG".) The compact syntax is often indicated as RNC and the XML syntax is often indicated as RNG.

XSD (XML Schema Definition), from the World Wide Web Consortium (W3C), is an alternative to the RELAX-NG syntax. It cannot express as many situations as the RELAX-NG syntax, but we have created the PreTeXt schema in a way that `trang` can convert to an XSD specification without introducing any more-permissive "approximations." But the XSD version seems to have a few small inaccuracies, and in particular should not be used for validation. That said, `schema/pretext.xsd` may be useful for tools (e.g. editors) that require it.

The files `pretext.xml`, `pretext.rnc`, `pretext.rng`, and `pretext.xsd` are all provided in the `schema` directory under revision control, and are updated by the `schema/build.sh` script when changes are made to `pretext.xml`. So as an author, you do not need to install or run `trang` and should just link to the (continually updated) copies in your `mathbook` directory.

We once provided a **document type definition** (DTD) as a description of the PreTeXt vocabulary. Mitch Keller wrote an excellent initial version of this chapter to help others understand similar principles in the context of the DTD. However, the DTD was not sufficiently flexible to handle elements that behave differently in different locations, such as an `<introduction>` to a `<chapter>` versus an `<introduction>` to an `<exercisegroup>`. As further evidence, `trang` will knowingly refuse to convert the PreTeXt schema to a DTD since the DTD syntax is not sufficiently expressive to describe PreTeXt.

If you are interested in conversions, more tools can be found at `relaxng.org` and we have information on installation in Appendix F. We would be pleased to learn more about authors' experiences with other converters.

5.4 Reading RELAX-NG Compact Syntax

The compact syntax might remind you of Java or C++ syntax. We do not provide a tutorial here, but do provide some hints on the various delimiters and special symbols, which may make it fairly easy to see your way through to the meaning. The fundamental object of the schema is a **pattern**, and **named patterns** can be reused in order to reuse common patterns and modularize the description. One approach to validation is to remove portions of your source that match required patterns until only optional material remains. Notice that if you were to chase your way through substituting the named patterns with their employment, you would have a single (large) pattern which every possible PreTeXt document would match, and by definition an XML document that does not match is not PreTeXt. (OK, that is a slight exaggeration, see Section 5.2.)

element foo { }	Define `<foo>` and children
attribute bar { }	Define `@bar` and values
text	Sequence of characters (any length)
mixed { }	Mixed-content, characters interleaved in pattern
	Exactly one (required)
+	One or more (required)
*	Zero or more (optional)
?	Zero or one (optional)
,	Sequence, in prescribed order
|	Choice
&	Sequence, any order
( )	Grouping
=	Define named pattern
|=	Accumulate named pattern as a choice

Table 5.4.1: RELAX-NG Compact Syntax Summary

5.5 Validation

We cannot stress enough the importance of validating your source early and frequently. Error conditions and messages can be built into processing (we have some anyway), but they are much better accomodated by tools built for just this purpose. If your processing with `xsltproc` suddenly fails, or if chunks of your content fail to materialize, it is highly likely that a validation check will tell you where the problem lies. If you integrate regular checks into your workflow, then sudden problems can be traced to recent changes. (Perhaps paired with using `git bisect`, in the extreme. You are using revision control, aren't you?)

We use `jing` for the first step, RELAX-NG validation. This is an open source companion to the `trang` converter described above. As a Java program, it should be cross-platform. It is also packaged for Debian and Ubuntu Linux. It provides messages keyed to the line number and character number of your source, and the messages are very clear and informative. See notes on installation in Appendix F. We would be pleased to learn more about authors' experiences with other validators.

You might get a lot of error messages the first time you use `jing`. If so, it might be that many of them are the same situation. If you pipe the output of `jing` through `sort -k 2` then the output will group similar messages together.

If you use `xsltproc` as your XSLT processor, then you likely automatically also have the `xmllint` program, which will perform validation with RELAX-NG schema. Our experience is that it bails out at the first warning, or at least does not process the remainder of the current subtree, and that the error messages are often very misleading. We will not support questions about validation based on output from `xmllint`.

The second step is easier, since it is an XSL transform. In other words, it is just another stylesheet, which you run against your source, with a processor like `xsltproc`. This stylesheet encoding Schematron rules is unique to PreTeXt and will report exceptions that are too difficult to express with RELAX-NG. So validation is two complementary steps. See Appendix F for the exact syntax for using this stylesheet.

5.6 Schema Browser

We use the DocFlex/XML XML Schema Documentation Generator to automatically produce hyperlinked documentation of the schema as part of the online documentation at the PreTeXt website, located at mathbook.pugetsound.edu/doc/schema/. Because this is produced from the XSD version of the schema, it will reproduce the small inaccuracies mentioned above. But it is still a very convenient and informative way to explore the schema, or use it for reference. If you know of a similar tool of the same quality, but which documents RELAX-NG schema, a report would be greatly appreciated.

Do not be intimidated by the list of roughly 300 elements in the left navigation panel. Many are configuration options, many are special-purpose, and many you will never use. (Someday we will do lexical analysis on a substantial range of PreTeXt texts to see just which elements do get used most frequently). Instead, scroll down to the 70 or so "Element Groups". These are thematic bundles of related elements, named to help you locate them later. The right panel will list the elements near the top as part of the "Complex Content Model." Just below you will see the "Known Usage Locations" which list places every element in the group may appear. Similarly, if you explore a particular element, you can see the pattern describing the attributes and elements allowed as children, and lists of the possible parents.

Elements which have complicated restrictions handled by Schematron rules will have some written documentation to this effect as part of their page within the schema browser.

The XML Schema Documentation Generator is a commercial program, but provides a free license allowing minimal customization and embeddeding promotion in the output. (Thank-you, Filigris Works!) In particular, as an author you should not need to install or use this program.

5.7 Editor Support for Schema

We collect summary information about editors that make use of schema. See Appendix E for more specific information, links, etc..

Editor	Formats	Notes
emacs	RNC	Schema-sensitive editing, open source
XML Copy Editor	RNG, XSD	Validation, "tag completion", open source
oXygen	RNC, RNG	Validation and completion, commercial

Table 5.7.1: Schema Support in Editors

Chapter 6

(*) Topics

This long chapter provides the main documentation for a variety of the features of PreTeXt. Some sections are just stubs and need to be written. Requests for sections to prioritize are welcome, though some sections are waiting for features to stabilize.

6.1 Paragraphs

Much of your writing will happen in paragraphs, delimited by the simple tag, `<p>`. You are reading one right now. They are a basic building block of divisions, and also a basic building block of other structures. For example, an ordered list, `<ol>`, contains a sequence of list items, `<li>`, and a typical list item might be a sequence of paragraphs. (Do not confuse this element with the anomalous `<paragraphs>` subdivision, Section 6.7).

Paragraphs are a choke-point of sorts. Many tags can *only* be used within paragraphs, and many others *cannot* be used within paragraphs. Notice too, that we do a certain amount of manipulation of whitespace in a paragraph, in ways that you may not even notice.

The following subsections together contain allowed, or encouraged, markup within a paragraph. Many of these may be used in captions and titles, but some of the more complicated constructions (which appear later here) cannot be used in captions or titles.

Note that some of this may appear to be overkill, and if you choose not to use it, you may even have success for a while with one of the output formats. But if you wish to ever produce multiple outputs, then the following is necessary. For example, a plain octothorpe (hash), #, will migrate just fine to HTML output, but will cause fatal errors in LaTeX output, and can also cause problems in output formats that employ Markdown. JSON is another component of some output formats which can be problematic.

We will say it again. PreTeXt is a markup language, and our various output formats (LaTeX, HTML, EPUB, Jupyter notebooks) in turn employ markup languages. These use different escape characters and give different characters special meanings. Our job is to insulate you from this variety, but it requires that you use markup in places where you might "normally" just press a key on your keyboard. The descriptions below will contain more specific information.

One more comment: typewriters, computer keyboards, and the ASCII character set limit the full range of characters that typographers and printers have used historically. A case in point is the hyphen, which is a single key on a keyboard. However, there are at least three common dashes of differing lengths (hyphen, en dash, and em dash), and in the context of mathematics or a computer program, the hyphen might be the binary operation of subtraction or the unary operation of negation. PreTeXt will help you navigate this complexity, but you will want to use keyboard characters or markup appropriately. So if you care about communicating clearly, and making your writing easy for a reader to use, absorb the details that follow and the philosophy they implement.

We begin with some simple "grouping" elements which contain several excellent examples of the importance and utility of careful markup. There is a plethora of empty tags for individual characters, and these are very important (see Subsection 6.1.4). We defer them to the end of this section, since they are not

as instructive, but do not think this means they are an afterthought. They can be extremely critical for successful conversions. Also do not miss Best Practice 6.1.1 in the conclusion of this section.

6.1.1 Simple Markup in Paragraphs

Beyond empty tags that translate to various characters, there are relatively simple tags that can call attention to various *portions* of a sentence, or generate more complicated constructions than described above.

Most, if not all, of the markup in this subsection may also be used within titles and captions, though they might lose some of their features when used in a title, especially when the title is duplicated in other contexts, such as a Table of Contents.

<q>, quotes, "group". This is the first of several grouping tags, using characters with left and right variants (see the individual characters in isolation above), and some of the most common markup in your writing. Presentation uses double quote marks that are **smart quotes**, meaning that they look different in their opening and closing variants. With output based on HTML, you may use your keyboard's double quote mark instead and get acceptable output, but your LaTeX will look wrong (plus you will look like you are a beginner with LaTeX) and outputs based on JSON will be *totally broken*. Futhermore, if you use the <q> your HTML output will look better (typographically) than if you do not use it. (See <blockquote> for extensive runs of quoted text that can stand alone, and which can carry an attribution.)

<sq>, single quotes, 'group'. Perhaps less-often used than <q>, so a couple more characters to type. Presentation is paired single-quotes, opening and closing. Read the discussion above about double-quotes if you have not already, this tag is entirely similar.

<braces>, braces, aka curly brackets, {group}. Left and right braces to enclose a phrase. This is not for creating a set in mathematics, use the appropriate mathematics tag and syntax for that.

<angles>, angle brackets, ⟨group⟩. Left and right angle brackets to enclose a phrase. This is not for creating a set of generators in mathematics, use the appropriate mathematics tag and syntax for that. Note also that the characters used here are distinct from the inequalities, <less> and <greater> ($<$ and $>$).

<brackets>, square brackets, [group]. Left and right square brackets to enclose a phrase. This is not for grouping expressions in mathematics, use the appropriate mathematics tag and syntax for that.

<dblbrackets>, double square brackets, ⟦group⟧. Double left and right square brackets to enclose a phrase. This is not for grouping expressions in mathematics, use the appropriate mathematics tag and syntax for that. These are used in the analysis of texts to note various restorations or deletions. Inquire if you feel there should be more semantic markup for this purpose.

, emphasis, *important*. Use this element to surround characters in a phrase that is to be emhasized. This will typically be rendered in italics, though this choice is left to the implementation of a particular conversion. See also, <alert>.

If you are new to using a markup language, this is a place to stop and think. As a PreTeXt author you are never able to say, "I want this text to appear in italics." Rather, you specify that certain text has a certain purpose or meaning. Emphasis is a way of *calling attention* to a portion of a sentence or paragraph. A font change (to italic) is a common and effective device. But a particular format might have a better, or different, way to achive this. Perhaps in an electronic format, the letters are animated and dance up and down. (Just kidding. But you may be reminded of frequent blinking text in the early days of web design, supported by a non-standard <blink> element.) Seriously, now would be a good time to review Section 1.1.

<alert>, alert, *critical*. Use this to heavily emphasize something to a greater degree than just emphasis. Maybe think of it as ***SHOUTING***. Bold italic font, or a bright color, or both, would be normal choices for presentation. Overuse of this tag will dilute its effectiveness.

<term>, terminology, larvae. Use this to identify a word or phrase that is being defined, in contrast to actually using a structured <definition>. Typical presentation is a bold font. Caution: the use of this tag is to communicate a defined term and converters may make use of this interpretation, given the importance of definitions in scholarly work. It would be considered **tag abuse** to use this tag to simply bold a word or phrase for some other reason, perhaps as an alternative to or <alert>.

<foreign>, foreign words, idioms, phrases, *Hola*. This tag is used to identify words or phrases from a language other than the main one used for the overall document. It is best practice to use a @xml:lang attribute to identify the language, since this will assist screen readers and hyphenation algorithms. We may also recognize the need for a different script (font). Usual presentation is italics for languages using a Latin script. This should not be used for entire paragraphs as a way of assisting with a translation of an entire document.

Note that when we use italics for emphasis *and* to point out foreign words or phrases, there is a loss of information in our output. In other words, we can no longer reliably (in an automated way) convert our output back to equivalent PreTeXt source from its visual representations. *C'est la vie*. See Section 1.1 again.

<pubtitle>, <articletitle>, titles of books and articles. These provide the ability to typographically distinguish the title of another work, and are not a replacement for careful bibliographies and citations. Use <pubtitle> for longer, complete works, such as books, plays, or entire websites. Use <articletitle> for shorter, component works, such as a chapter of a book, a research article, or a single webpage.

Presentation for a longer work will be italics or an oblique (slanted) font, and for a shorter work, the title will simply be quoted.

<abbr>, <init>, <acro>, abbreviation, initialism, acronym, Mr., XML, SCUBA. An abbreviation is a condensed or shortened version of some word or phrase, such as MR. for "Mister." Converters should take care with periods (full stop) inside an <abbr> as distinguished from the end of a sentence (which may not always be clear given the absence of a tag delimiting sentences). An initialism is an abbreviation read as a sequence of letters, often the first letter of words in a phrase, such as HTML for "HyperText Markup Language." An acronym is much like an initialism, but the letters are read as a pronouncable word (which sometimes actually enters the language as a word, such as "radar" which began as RAdio Detection And Ranging). An example is SCUBA which stands for "Self-Contained Underwater Breathing Apparatus." Initialisms and acronyms may be presented in a small-capitals font or as regular capitals reduced in size.

<delete>, <insert>, <stale>, editing assistance, ~~gone~~, new, ~~old~~. These denote portions of a text that is being changed in some way, presumably as part of an editing process. Conceivably, they could be managed by some other tool acting on your source. Stale text is that which is slated for removal eventually, but is left in place so that it may be consulted. There is no support presently for actually deleting or incorporating text, though that would be a reasonable feature request.

Red and green, for leaving and entering, are natural choices for presentation. But in consideration of those readers who cannot always distinguish different colors, other devices, such as strikethrough or underlining, should also be employed.

<tag>, <tage>, <attr>, tag, empty tag, attribute, <section>, <hash/>, @width. These are PreTeXt tags for when we write about PreTeXt and need to discuss tags, empty tags, and attributes. Given how we design PreTeXt tags the content of these elements should only be the 26 lower-case letters and a dash/hyphen. These should render in ways that make the three types of language elements obvious without much further discussion. Just a bit self-serving, but not unjustified.

<taxon>, scientific names, *Escherichia coli*. This element may surround a full scientific name, resulting in presentation in italics. There are subelements <genus> and <species> which may be used to delineate

those components.

A @ncbi attribute on <taxon> accepts an identiier from the National Center for Biotechnology Information. Feature requests for ways to make this more useful are welcome.

<fn>, footnotes. A footnote can be inserted in a paragraph and a mark will be left behind. Where the content of the footnote goes depends on the capabilities of the output format. Because a footnote allows you to begin a new piece of text *anywhere*, it can be difficult to handle technically. For this reason it is banned from places like titles and its possible content is limited (for openers, no paragraphs).

A footnote is the farthest thing from structured writing that we can think of. It can go anywhere. Resist the temptation to use it, and your writing will improve. We frequently entertain the thought of making footnotes impossible in PreTeXt. See the <aside> for a possible alternative.

<m>, mathematics, $x^2 + y^2$**.** Simple, inline expressions using mathematical notation may be used in paragraphs, and in titles and captions. The syntax is LaTeX. See Section 6.8 for full details.

<c>, code, verbatim text, literal text, import. Short bursts of raw, or verbatim, text can be wrapped in the <c> element. Strictly speaking, "code" is a misnomer, as the text may be anything you need to communicate exactly as one would type it at a keyboard or as input to a computer program. Anything longer than a handful of characters, or needing accurate line breaks should consider the <cd>, <pre>, <program> or <console> tags. Presentation is normally a monospaced sans serif font, perhaps of a slightly heavier weight, and designed for the job with features such as unambiguous zeros (versus the letter 'oh'). See Section 6.5 for details.

<email>, email address, nobody@example.com**.** Very similar to the <c> tag, this may be used to get a monospace presentation of an email address, possibly as an active link in some formats.

6.1.2 Cross-References and Paragraphs

There are several devices for creating cross-references. Generally, these are unwise (or banned) in titles, and if allowed may be inactive in certain portions of an electronic output format (such as when migrating to a Table of Contents).

<url>, linking external resources. This is a cross-reference *to* some item separate and distinct for your document.

A Uniform Resource Locator (URL) is, loosely speaking, an Internet address for some item. Presentation depends on the capabilites of the output format to serve the resource. There is a mandatory atrribute, @href, that contains the full URL, including a protocol (such as http://). Used as an empty tag, the visual text will be the exact contents of the @href attribute. So, http://www.example.com can be achieved with

```
<url href="http://www.example.com"/>
```

You may also wish to provide some text other than the actual URL, which you can specify as the content of the <url> element. For example, IANA Test Site can be achieved with

```
<url href="http://www.example.com">IANA Test Site</url>
```

In order to have a URL occur in print output in a useful way, and in electronic output in an active way, I often shorten the full URL in the visual portion and mark it up as verbatim text. So illustrating again, we get example.com from

```
<url href="http://www.example.com"><c>www.example.com</c></url>
```

Notice the necessity and/or desirability of marking the text in a way that distinguishes it as literal text.

Note also that this tag is meant for *external* resources, so see the <xref> element (below) or Section 6.6 for ways to link internally (i.e. within your document).

<xref>, cross-references. This is a cross-reference *to* some item contained within your document.

Extensive and intuitive capabilities for cross-referencing are a primary feature of PreTeXt. Typical use is an empty tag with the `@ref` attribute specifying the value of an `@xml:id` on the **target** of the cross-reference. This should work easily without much more instruction, but familiarize yourself with the details in Section 6.6 to get the most out of some the available options.

<idx>, index target. This indicates that the containing structure (theorem, example, etc.), or top-level paragraph, should be the *target* of an entry of the index (a special sort of cross-reference). See Section 3.20 and Section 6.16 for general details.

<notation>, index target. This indicates that the containing definition, or top-level paragraph, should be the *target* of an entry of the list of notation (a special sort of cross-reference). See Section 3.20 and Section 6.16 for general details.

6.1.3 Structured Markup in Paragraphs

There are three categories of items which typically are structured further, and which are almost entirely restricted to appearing in a paragraph. Given their complexity, details are covered in other sections of this guide.

Lists. With only one major exception (and three minor ones), a list *must* appear within a paragraph. See Section 3.7 for an introduction and Section 6.9 for precise details.

Display Mathematics. Displayed mathematics, which is a single equation or a sequence of (aligned) equations, can only be placed within a paragraph. The relevant tags are `<me>`, `<men>`, `<md>`, and `<mdn>`, with the latter two necessarily structured with `<mrow>` elements. See Section 3.5 for an introduction and Section 6.8 for precise details.

Display Verbatim. The `<cd>` tag, by analogy with the `<md>` tag for displayed mathematics, may be used to display one or more lines of verbatim text (such as a series of commands), possibly structured with the `<cline>` tag. See Section 3.14 for an introduction and Section 6.5 for precise details.

This should not be confused with the `<pre>`, `<console>`, or `<program>` tags, which have slightly different uses, and all of which must be used *outside* of a paragraph.

6.1.4 Characters in Paragraphs

<nbsp/>, non-breaking space. A space, but which ties two words together and discourages a line break when formatted, such as `Summer<nbsp />1967`. This can also be used to discourage a period in an abbreviation from being interpreted as the end of a sentence, such as `C.R.<nbsp />Darwin`.

<ndash/>, –, en dash. A dash, the width of a lowercase 'n', or exactly half the width of the em dash. This is typically used to express a range, such as `1955<ndash />1975`, with no intervening spaces. It is often expressed as two hyphens when typed. Bringhurst suggests an ndash surrounded by spaces – thusly – when setting off phrases.

<mdash/>, —, em dash. A dash, the width of a lowercase 'm', or twice the width of the en dash. This is typically used to express a secondary part of a phrase, much like the use of a semi-colon or parentheses.

<ampersand/>, &. This is the symbol often read as "and", a stylised contraction of "et". Be careful, ampersands in mathematics and in verbatim text (code) are implemented differently. This version is for use with "normal" text. This is the escape character for XML and has special meanings in LaTeX.

ALWAYS use this empty element in normal text, or no processing of any kind will succeed. Review Section 3.13 for use within mathematics or verbatim text. See also Subsection 6.8.4 and the introduction to Section 6.5.

<less/>, <greater/>, <, >, less than, greater than. If you need these symbols in "normal" text (which should be unlikely), this is the way to get them. A bare '<' will totally confuse the XML processor, since, as you know now they have a special mening in XML syntax. Mathematics and verbatim text have their own variants. Note that these are the keyboard characters commonly used for inequalities in text, and are different from the grouping characters <langle/> and <rangle/> discussed below.

ALWAYS use these empty elements in normal text, or no processing of any kind will succeed. Review Section 3.13 for use within mathematics or verbatim text. See also Subsection 6.8.4 and the introduction to Section 6.5.

<hash/>, #, octothorpe, numeral sign, pound avoirdupois, hash. This character has special meaning in LaTeX, and in Markdown, so consistently using this empty element for normal text is critical for successful conversion to popular outputs, and so is a good practice.

<dollar/>, $, dollar sign. This character has special meaning in LaTeX syntax for mathematics, so consistently using this empty element for normal text is critical for successful conversion to every output, and so should be used *ALWAYS*.

<percent/>, %, percent, per centum. This character has special meaning in LaTeX, so consistently using this empty element for normal text is critical for successful conversion to popular outputs, and so is a good practice.

<circumflex/>, ^, circumflex, caret. This character has special meaning in LaTeX, so consistently using this empty element for normal text is critical for successful conversion to popular outputs, and so is a good practice.

<underscore/>, _, underscore. This character has special meaning in LaTeX, and in Markdown, so consistently using this empty element for normal text is critical for successful conversion to popular outputs, and so is a good practice.

<lbrace/>, <rbrace/>, {, }, left brace, right brace. These characters have special meaning in LaTeX, and in JSON, so consistently using these empty elements for normal text is critical for successful conversion to popular outputs, and so is a good practice.

<tilde/>, ~, tilde. This character has special meaning in LaTeX, so consistently using this empty element for normal text is critical for successful conversion to popular outputs, and so is a good practice.

<backslash/>, \, backslash. This is the escape character for LaTeX, Markdown, *and* JSON! So consistently using this empty element for normal text is critical for successful conversion to almost every popular output, and so is recommended *ALWAYS*.

<lq/>, <rq/>, “, ”, left and right double-quotes. These are the characters in isolation. If you are using them in pairs, see the <q> tag below for a matched pair. Note that LaTeX syntax expresses quote marks in a very complicated way, and typographically a left quote mark is different from a right quote mark, so it is never good practice to use the keyboard versions.

<lsq/>, <rsq/>, ‘, ’, left and right single-quotes. These are the characters in isolation. If you are using them in pairs, see the <q> tag below for a matched pair. Note that LaTeX syntax expresses quote marks in a very complicated way, and typographically a left quote mark is different from a right quote mark, so it is never good practice to use the keyboard versions. In particular, note that the <lsq/> character is not the same as the "backtick" of various markup languages.

<ellipsis/>, ..., ellipsis. Typically three low dots with no intervening space, to indicate a continuation. This will always perform better than three consecutive periods.

<asterisk/>, ∗, asterisk, star. This is the character, and not the raised mark used to indicate a footnote or endnote. It has a special meaning in Markdown.

<backtick/>, `, backtick, accent grave. This is the keyboard character that looks like a left single quote. It is not a quote mark (use <lsq> for that character). It is often used to modify other characters, as an accent. But that is not the use of this empty element either. This is the keyboard character, which is sometimes used in other markup languages. So, again, you could do fine pressing the key, but LaTeX might turn it into a left quote mark, and it might cause confusion when Markdown is employed.

<slash/>, /, slash, forward slash. This is the character used to separate words/information/text. It is not to be confused with the <solidus/> (or virgule) used to form a fraction in normal text. You should be able to reliably use the keyboard character / instead. We include the markup version if making the distinction is important for your project.

<midpoint/>, ·, midpoint. A small centered (vertically) dot, which can be used to separate pieces of information, especially in displayed text (i.e. outside of paragraphs). Not to be confused with a bullet preceding a list item, or multiplication in mathematics.

<swungdash/>, ∼, swung dash. Another decorative separator, not to be confused with the keyboard tilde character.

<permille/>, ‰, per mille. Like per cent, but now a number expressed as its product with 1000 (rather than with 100).

<pilcrow/>, ¶, pilcrow, paragraph mark. Mark used historically to indicate the start of an internal paragraph, and in a more modern use, to indicate a permalink.

<section-mark/>, §, section mark. Used to prefix the number of a section, or other division. (So the word section is being used generically here.)

<copyright/>, ©, copyright. The symbol used in publishing, legal, or business contexts. For a PreTeXt project, copyright information can be specified within the <colophon> portion of the <frontmatter>.

<trademark/>, ™, trademark. The symbol used in legal or business contexts.

<registered/>, ®, registered. The symbol used in legal or business contexts.

<today/>, <timeofday/>, November 26, 2018, 21:05:46 (-08:00). Values at the time of XML processing. Useful for marking drafts or other frequently revised material such as online versions.

<tex/>, <latex/>, <pretext/>, <webwork/>, TeX, LaTeX, PreTeXt, WeBWorK. Conveniences for frequently-mentioned accessories to PreTeXt.

<ie/>, <eg/>, <circa/>, <versus>, <etc/>, i.e., e.g., c., vs., etc.. A small collection of frequently-used Latin abbreviations, with attempts to handle the tricky periods wisely in LaTeX output. Strictly speaking BC is not Latin, but we include it for completeness. Tags are always lowercase, no punctuation, usually two letters.

Tag	Realization	Meaning
ad	AD	*anno Domini*, in the year of the Lord
am	A.M.	*ante meridiem*, before midday
bc	BC	English, before Christ
circa	c.	*circa*, about
eg	e.g.	*exempli gratia*, for example
etal	et al.	*et alia*, and others
etc	etc.	*et caetera*, and the rest
ie	i.e.	*id est*, in other words
nb	N.B.	*nota bene*, note well
pm	P.M.	*post meridiem*, after midday
ps	P.S.	*post scriptum*, after what has been written
vs	vs.	*versus*, against
viz	viz.	*videlicet*, namely

<fillin/>, ____, fill-in blank. A "fill in the blank" blank. While meant for use with normal text, it may also be used within mathematics contexts. The @characters attribute may be used to hint at how long the line will be, should the default value of 10 be too long or too short. Here we have set @characters to the value 5.

<times/>, ×, times, multiplication. For simple arithmetic expressions in text, this symbol may be used. Or it may be used to specify dimensions, as in "I bought a 2×4 at the lumber yard."

<solidus/>, /, solidus, virgule, fraction bar. For simple arithmetic expressions in text, this symbol may be used to form a fraction. It shoulds appear to have a significantly shallower slope than the forward slash, /.

SI Units. *Système international (d'unités)* (International System of Units) is a system of measurement used universally in science. PreTeXt has comprehensive support for this system and its notation and abbreviations. See Section 3.24 for a short introduction and Section 6.22 for detailed descriptions of the relevant elements and their use.

𝄪, ♯, ♮, ♭, 𝄫, music notation. Notes, chords, and other notation may appear within sentences as part of a dicussion. See Section 6.21 for detailed descriptions of the relevant elements.

Best Practice 6.1.1 Understand the Importance of Careful Markup. There is a lot of detailed information in this section. Much of it is critically important, some is superfluous. If you are new to thinking in terms of markup (rather than WYSIWYG tools), it might be overwhelming, a lot to digest, and hard to separate the wheat from the chaff. Careful here means using the necessary markup, not using it for other purposes different than its intent (**tag abuse**), planning ahead for different output formats, but not becoming a slave to over-doing it.

So come back here often for a re-read. And keep in mind that PreTeXt is designed around principles (List 1.1.1), and that it is markup (Item 1.1.1:1) which enables multiple outputs (Item 1.1.1:3) and effective and beautiful online versions (Item 1.1.1:6).

6.2 (*) Exercises, Inline and Divisional

6.3 (*) Worksheets

6.4 (*) Lists of Works Cited (References)

6.5 Verbatim and Literal Text

This section expands on parts of Section 3.14. For descriptions of more involved uses, such as program listings and console sessions, see Section 6.13.

The tags described here contain *only raw characters*. By that we typically mean the first 128 characters of the ASCII code. Unicode characters are likely to migrate to HTML output just fine, but results for LaTeX output will be variable. The restriction to character data has two consequences. First, any markup mistakenly included will have its content silently ignored and dropped. Second, you need to observe the rules on special characters and escaped characters for XML for literal text, which are mercifully simple.

> In your source, use `&` for an &, and use `<` and `>` for < and >. Otherwise, every other ASCII character will render faithfully across all possible formats.

6.5.1 Short, Inline, Verbatim Text

The <c> tag is a mnemonic for "code", but is really meant to be any chunk of literal characters that you want to emphasise that way. So a "typewriter" font of fixed-width characters would be a typical presentation. It is meant for use within a sentence or caption ("inline") so its use is limited to those situations, and others that are similar, such as a title or a cell of a table. Typically these pieces of text do not hyphenate words, and so can lead to spillover into a right margin. In these situations, consider options below for longer pieces of text.

6.5.2 Longer, Inline, Verbatim Text

For longer pieces of verbatim text, use the <cd> tag, which is short for "code display", analogous to the <md> for mathematics. It is used within sentences of a paragraph and will be presented with a vertical break above and below, but without interrupting the paragraph. Because of the display presentation, it cannot be used other places, such as a <title>, where a vertical gap is not appropriate. All of the previous discussion about special characters applies for this tag.

You have two options in use. You may author inline with the rest of a sentence, with no extra newlines or whitespace before, after, or within the content. The result will be a single displayed line.

Or you may structure the content using one, or more, of the <cline> tag, which is meant to be similar to the <line> tag used elsewhere. You should still take care to not place any extra whitespace before or after the <cd> element, but inbetween the <cline> you may use as much visual formatting of your source as you wish, especially if you like your source to mirror your output.

6.5.3 Blocks of Verbatim Text

If you want to isolate large chunks of verbatim text outside of paragraphs, the <pre> tag is the one to use. It can be used as a peer of paragraphs (and other structures) as a child of a division, or it can be placed into a <listing> to receive a caption, title and number.

You can structure the contents with <cline> in exactly the same manner as for <cd>. But you may find this tedious. Instead, you can make the content of <pre> a sequence of lines separated by newlines. So that you can preserve the indentation of your source, the line closest to the left margin is taken to actually be the left margin, and a corresponding amount of leading whitespace will be removed from every line. This will work well if you recognize two caveats. First, results will be unpredictable if you mix spaces and tabs for indentation. Sticking with spaces is best. Second, if your first characters of content immediately follow

the <pre> tag then there is no leading whitespace and it is as if that line is already at the left margin. Then subsequent indentation may seem too severe to you.

As previously mentioned, Section 6.13 discusses the <console> and <program> tags which are more specific, and hence more capable. Review the possibilities before you decide between <pre>, <console>, and <program>.

6.6 Cross-References and Citations

When you read a novel, you would likely read it cover-to-cover (in one sitting?) and then put it away and never read it again. But for a textbook, you may read cover-to-cover, but you may also frequently skip around, especially at exam time. And once read, it might become a reference work for you, since you know it so well. So years later you might come back looking for something. Cross-references help with all this, so use them liberally. Recognize that an index is really just a specialized way to provide an abundance of cross-references.

6.6.1 Creating a Cross-Reference

It is a two-step process to make a cross-reference.

1. Put an @xml:id attribute on any element you think you might want to reference later. Be organized about the values of these attributes, and in particular do not number them, as this has no place in your source, and you do not want to maintain the changing numbers over the life of your project. See the advice in Section 3.3 about banned characters. Some elements do not accept this attribute because the element has nothing to identify it (no number, no title). Typically these are **containers** such as <sidebyside>, <statement>, or <ol>. In these cases, put the attribute on the closest enclosing element.

2. To make a cross reference, you create an <xref/> element with a @ref attribute with the same value as @xml:id attribute on the element you want to reference.

Simple. It is meant to be, so you can use it liberally. But we also know authors want some flexibility.

Best Practice 6.6.1 Use @xml:id Frequently. Use the @xml:id attribute liberally, on any object you might want to reference later, and on every division. It is easier to do as you author and will be very valuable later. (Trust us on this one.) Develop a system so you can recall them predictably, but keep them readable. Don't use numbers, they will change. Then make frequent cross-references. They are relatively easy for you and will be incredibly useful for each and every one of your many readers, over and over and over again.

6.6.2 Text of a Cross-Reference

By default, a cross-reference will visually be text like Theorem 5.2. Depending on your output format, it may have various devices to help you locate that theorem. Maybe a page number, or it is a hyperlink, or the whole theorem is the content of a knowl. You can change the default look of cross-references by setting the @text attribute in docinfo/cross-references. But you can also change the visual appearance of a cross-reference on a case-by-case basis. Add a @text attribute to your <xref/> element to override the document-wide setting. The first column of this table lists the possible attribute values, either document-wide, or on a per-cross-reference basis. The second column has live cross-references to a section of an earlier chapter (that is far away). The third column has live cross-references to another section of this chapter (which is close by). For the fourth column, we have placed content ("Extra") into the <xref> element as an override of, or addition to, some of the text for the cross-references of the preceding column. Study the table and then read some more explanation following. Note that type-global is the default.

@text	Far Away	Close By	With Content
type-local	Section 4	Section 7	Extra 7
type-global	Section 3.4	Section 6.7	Extra 6.7
type-hybrid	Section 3.4	Section 7	Extra 7
local	4	7	Extra 7
global	3.4	6.7	Extra 6.7
hybrid	3.4	7	Extra 7
phrase-global	Section 4 of Chapter 3	Section 7 of Chapter 6	Warning
phrase-hybrid	Section 4 of Chapter 3	Section 7	Warning
title	Titles	Divisions	Extra

Table 6.6.2: Cross-reference visual text styles

Note that local/global describes the uniqueness of the number (and is affected by your choice of numbering schemes), while type refers to an automatic prefix of the number. The text of the type will vary according to the document's language. If a cross-reference and its target are close to each other, a number like 5.8.2.4 might be overkill, when just a 4 would suffice. A hybrid scheme will use the shorter number whenever it makes sense. Finally, there are two phrase schemes, and it should be clear what text title will produce (though realize there must be a title for the object, possibly a default provided by PreTeXt).

You can also override the text used in a cross-reference link. You do this by providing content to the <xref> element. In other words, do not use an empty tag, but put some (simple) text in the element. Generally, this additional text becomes a prefix of a number, a replacement of a type, or a replacement of a title. It is better to use these overrides, since in electronic formats, the text of the override will be incorporated into the "clickable" portion of the resulting link, making a larger item to hit.

Here are careful examples, where we have provided the content "Division" in the live examples. The list is not exhaustive. Note that overriding a title provides a way to provide any text you would like.

@text	Example
'global'	Division 6.6
'type-global'	Division 6.6
'title'	Division

Table 6.6.3: Cross-reference text overrides

6.6.3 Variations

There are some variant uses for the <xref> tag.

- Replace @ref by @provisional and give it a value with some simple text like subsection on eagle habitat and you will get reminders that once you write this future subsection you need to link it in right here. This is just a convenience for authors during the early stages of a writing project.
- Replace @ref by the pair @first and @last. Provide attribute values just as you would for @ref. The code will check that the targets have the exact same tag, and that the order is correct. You will get a link that looks like a range, separated by an en dash. As a link, only @first will be used for the linking mechanism (i.e., one link is generated, not two). Experiment with @text and content overrides.
- The @ref attribute may be a list of @xml:id, separated by commas or by spaces. Then you will get back a list of cross-references. This is meant for a list of citations, producing a look like [5, 9, 22], but it makes no restriction to this case. Use it generally, but it is unlikely to get any more capable. If you want a different list, just use multiple <xref> and format as you desire.

You can create many different combinations with the text options and the variants. Here is one example. You want to say Chapters 1–5. As a range you use the variant with two references. You would get "Chapter"

out front automatically with the type-global scheme, but a plural form makes more sense. So you use that text as an override. We could use either type-global or global to get the same text, and possibly type-hybrid depending on teh place where you built the cross-reference. So possibly, one of these schemes might be your document-wide setting and you do not need to specify it now. Here is what we just used:

```
<xref first="start-here" last="schema" text="global">Chapters</xref>
```

You can place a cross-reference into a <title> element, but a particular conversion is free to simply render it as text, and not as an active link.

Finally, there is fairly robust error-checking to protect against typographical errors in the attribute values that need to match up for all this to work. Also, there is a check that the @xml:id are unique. But all this checking happens at processing-time, not at the validation step. Any suggestions for improvements that make these checks even more robust are welcome.

6.6.4 Citations

Citations are just specialized cross-references to <biblio> elements, and so behave the same way as other cross-references. However they will always visually look like [23], and there is no notion of "type name."

6.6.5 Equations

Similar to citations, references to equations (<men> and <mrow> elements) will visually look like (4.2), where the type name is implied by the parentheses. Notice that you cannot cross-reference an <me> element (it has no number) or an <md> element (it is just a container, filled with <mrow> that you can target if you give them numbers). Consult Subsection 6.8.3 for details about controlling the numbering of equations within an <md> or <mdn> element.

6.6.6 Lists

Roughly, you can target a list item for a cross-reference, but not the list itself, since it is a container. See Subsection 6.9.4 for precise details about using list content as the target of a cross-reference. Note also, that an entire named list may be the target of a cross-reference, see Subsection 6.9.6. Here we concentrate on the text of the cross-reference itself.

First, note that if you cross-reference a list item of an anonymous list, there is a very real possibility that the number will be ambiguous, and there is no option for @text will save you, and never can be. See the middle column of Table 6.6.5 for the demonstration. We assiduously try to make it *impossible* to ever create ambiguous text for cross-references, especially in consideration of print output. Use the feature carefully.

Best Practice 6.6.4 Take Care Referencing Anonymous Lists. Cross-references to list items of anonymous lists can easily be ambiguous and then useless for readers of print. Keep such a cross-reference close to its target, ideally within the same list, and/or perhaps using additional, unambiguous clues about location (which you expect will survive later editing):

1. See Item 2 of this list.
2. See Item 1 in the list appearing in Best Practice 6.6.4.

The local option, discussed generally above, behaves differently for a cross-reference to a list item of an ordered list that is contained in a named list. As seen in the table just below, the local portion of the number is the part that comes from the list item, without the part that comes from the location of the <list> block.

@text	Named List	Anonymous List	With Content
type-local	Item B.c	Item 2.III	Extra B.c
type-global	Item 6.9.4:B.c	Item 2.III	Extra 6.9.4:B.c
type-hybrid	Item 6.9.4:B.c	Item 2.III	Extra 6.9.4:B.c
local	B.c	2.III	Extra B.c
global	6.9.4:B.c	2.III	Extra 6.9.4:B.c
hybrid	6.9.4:B.c	2.III	Extra 6.9.4:B.c
phrase-global	Item B.c of List 6.9.4	Warning	Warning
phrase-hybrid	Item B.c of List 6.9.4	Warning	Warning
title			Extra

Table 6.6.5: Cross-references to list items, visual text styles

The hybrid options employ a different definition of when the distance between a cross-reference and its target is close enough that the number can be shortend, without becoming ambiguous. When an <xref> and its target are part of the same <list>, then the common part of the number derived from the <list> is not needed. To illustrate we need to make a small <list> with cross-references contained within.

1. (type-global) Flowers are not Item 6.6.6:6.
2. (type-hybrid) Fish are not Item 6.
3. (hybrid) Bacteria are not 6.
4. (phrase-global) Fungi are not Item 6.
5. (phrase-hybrid) Trees are not Item 6.
6. Mammals.

List 6.6.6: Small test

Best Practice 6.6.7 Be Rational About Numbering Variations. With distinct numbering schemes for divisions, theorems, figures, equations, and citations, along with nine different text styles for a cross-reference, plus variants, per-cross-reference settings, and text overrides, there is a huge supply of combinations. Likely you can create some really ugly cross-references. Use the flexibility sensibly.

6.7 Divisions

A **division** (or more carefully, a **structural division**) is a structured component of a book or article that would be recognized by most any reader. They are essential to the organization of a PreTeXt project. Notice that we use the generic term division, since a <section> is just one example of a division.

Divisions are <book>, <article>, <part>, <chapter>, <section>, <subsection>, <subsubsection>, and <paragraphs>. Their use is fairly intuitive, though there are some restrictions, so please read on.

A <book> must contain at least one <part> or at least one <chapter>, which may contain <section>, <subsection>, and <subsubsection>. A <part> simply contains a sequence of <chapter> and functions in two user-selectable ways: structural (e.g. numbering will reset), or decorative (merely inserting a decorative page between two chapters and sectioning the Table of Contents).

An <article> is simpler and shorter than a book. It might be really simple and have no divisions at all, or it may have <section>s. It cannot have <chapter>s, as that would be a <book>. Within a <section>,

<subsection>s and <subsubsection>s may follow.

Divisions must nest properly and may not be skipped. So a <section> cannot contain a <chapter> and a <subsection> may not be contained in a <chapter> without an intervening <section>.

A division *must* contain a <title>, and may contain one or more index entries (see Section 6.16), which should appear before anything else. Any division may be unstructured, with just a sequence of **top-level content** such as paragraphs, figures, lists, theorems, etc. Or a division may be structured, and in this case it must follow a prescribed pattern. There may be a single, optional <introduction>, filled with top-level content, followed by a sequence of at least one of the appropriate divisions, ending with a single, optional <conclusion>, filled with top-level content. It is an error to begin with a run of top-level content inside a division and then begin to use divisions. (The solution is to make the initial content an <introduction> and/or one or several divisions.)

There are exceptions to the above. For one, <paragraphs> is an anomalous division, as a sort of lightweight sectioning command. It may appear in any division, at any location within a division, it may not be divided further (it is a leaf of the document tree), it never gets a number, and its title is formatted in a subsidiary way. I especially like to use this in a two- or three-page <article> that has no other divisions at all. Typical presentation has the title in bold, without much change in font size (if at all), inline with the first paragraph, and perhaps a bit of vertical space as it begins and ends. Despite the name, it may contain more than just paragaphs, so may contain any top-level-content that would go in any other division.

Two other anomalous divisions are <exercises> and <references>. These can be placed as a final component of any division from <chapter> on down. So for example, an <exercises> could be a peer of several <section>s, contained within a structured <chapter>, and in this case would behave similar to the other <section>s. Or an unstructured <chapter> may have a sequence of paragraphs, figures, examples, and the like, and conclude with a single <exercises>. Detail on allowed content and behavior are in Section 6.2 and Section 6.4, respectively.

6.8 Mathematics

As mentioned in the overview, Section 3.5, we use LaTeX syntax for mathematics. In order to allow for quality display in HTML, and other electronic formats, this limits us to the subsset of LaTeX supported by the very capable MathJax Javascript library. Generally this looks like the amsmath package maintained by the American Mathematical Society at their AMS-LaTeX page. For a complete and precise list of what MathJax supports, see the MathJax Supported LaTeX commands page.

6.8.1 Inline Mathematics

Use the <m> to place variables or very short expressions within a sentence of a paragraph, the content of a <title>, a <cell> of a table, a footnote, or other similar locations of sentence-like text. You can't cross-reference this text, nor make a knowl with it. Though you can typically cross-reference a containing element.

Do not use LaTeX-isms like \displaystyle to try to end-run the inline nature. It will just lead to poor results.

6.8.2 One-Line Display Mathematics

The <me> element can be used for longer expressions or a single equation. Typically you will get vertical separation above and below, and the contents will be centered. See below about concluding periods (and other punctuation), and alignment. The <men> variant will produce a numbered equation, and therefore with a provided @xml:id attribute, can be the target of a cross-reference (<xref>).

6.8.3 Multi-line Display Mathematics

We begin with a pure container, either <md> or <mdn>. The former numbers no lines, the latter numbers every line. Within the container, content, on a per-line basis, goes into a <mrow> element. You can think of

<mrow> as being very similar to <me>. If you are tempted to put a LaTeX \\ into an <mrow>, think twice.

On any given <mrow> you can place the @number attribute, with allowable values of yes and no. These will typically be used to override the behavior inherited by the container, but there is no harm if they are redundant. A given line of the display may be the target of a cross-reference, though the numbering flexibility means you can try (and fail) to target an unnumbered equation.

An <mrow> may have a @tag attribute in place of a @number attribute. This will create a "number" on the equation which is just a symbol. This is meant for situations where you do not want to use numbers, and the resulting cross-reference is "local." In other words, the <xref> and its target are not far apart, such as maybe within the same <example> or the same <proof>. Allowable values for the attribute are: star, dstar, tstar, dagger, ddagger, tdagger, hash, dhash, thash, maltese, dmaltese, tmaltese. These are the names of symbols, with prefixes where the prefix d means "double", and the prefix t means "triple". Cross-references to these tagged equations happens in the usual way and should behave as expected. See Section 3.3 and Section 6.6 for more on cross-references.

6.8.4 Special Characters

The LaTeX macros, \amp, \lt, and \gt are always available within these mathematics elements, so that you can avoid the special XML characters &, < and >. See Section 3.13 for this same information, but in the broader context of your entire document.

6.8.5 Text in Mathematics

Once in a while, you need a little bit of "regular" text within an expression and you do not want it to look like a product of a bunch of one-letter variables. Use the \text{} macro for these. Only. Other ways of switching out of math-mode and into some sort of "regular" text will appear inferior, and can raise errors in certain conversions.

- Do place surrounding spaces inside the \text{} macro.
- Do not place any mathematics inside the \text{} macro.
- Do not use the \mbox{} macro as a substitute.
- Do not use font-changing commands (e.g. \rm) as a substitute.

For example,

```
<me>f(x) = \begin{cases} x^2 \amp \text{if } x\gt 0\\
           -7 \amp \text{otherwise} \end{cases}</me>
```

produces

$$f(x) = \begin{cases} x^2 & \text{if } x > 0 \\ -7 & \text{otherwise} \end{cases}.$$

This example amply illustrates the use of macros for XML special characters (twice), appropriate use of the \text{} macro (twice), spaces in the \text{} macro (once), sentence-ending punctuation (see the source, the period is *not* inside the <me> element) and yes, we did think twice about the \\ (an exception to the rule).

6.8.6 Cross-References in Display Mathematics

A cross-reference is achieved with the <xref> element, see Section 3.3. You can place an <xref> inside a <mrow>, and remarkably, it will do the right thing. This is one of only two XML elements you can mix-in with LaTeX syntax. A typical use is to provide a justification or explanation for a step in a proof, derivation, or simplification. And it works best with alignment, see below.

6.8.7 Alignment in Display Mathematics

Displayed mathematics is implemented with the AMS-LaTeX `align` environment. Ampersands are used to control this, so use the `\amp` macro for these. The first ampersand in a line or row is an alignment point, typically a symbol, like an equality. The next ampersand is a column separator, then the next is an alignment point, then a column separator, then... The moral of the story is you should have n alignment points, with $n-1$ column separators, for a total of $2n-1$ ampersands—always an odd number.

For example,

```
<md>
  <mrow>A \amp = B \amp D \amp = E \amp \amp \text{Because}</mrow>
  <mrow>  \amp = C \amp   \amp = F \amp \amp <xref ref="txo" /></mrow>
</md>
```

produces

$$\begin{aligned} A &= B & D &= E & &\text{Because} \\ &= C & &= F & &\text{Table 6.6.3.} \end{aligned}$$

Sometimes you want several short equations on one line. Do not use `<me>`. Instead use a single `<mrow>` inside an `<md>`, and use alignment to spread them out evenly.

For multi-line display mathematics with no ampersands present, each line will be centered. This is implemented with the AMS-LaTeX `gather` environment.

You can fool the alignment behavior by hiding all your ampersands in macro definitions, so there is the optional `@alignment` attribute for the `<md>` or `<mdn>` element, in order to force the right kind of alignment. Allowable values are `gather`, `align`, and `alignat`. The latter is similar to `align`, but no space is automatically provided between columns. You can leave it that way, or explicitly add your own. For example, this allows you to precisely arrange individual terms of a system of linear equations, especially when terms with zero coefficients are omitted. When using the `alignat` option PreTeXt tries to count ampersands to see how many columns you intend, since LaTeX needs this number (we are not sure why). This detection can be fooled too, especially if you have something like a matrix with lots of ampersands for other purposes. So set the `@alignat-columns` attribute to the *number of intended columns*, if necessary.

6.8.8 Fill-In Blanks in Mathematics

The other mix-in XML element is `<fillin>` with an optional `@characters` attribute that takes an integer value. You will get a thin horizontal line, on the baseline, which can suggest to a reader that they should supply something within the surrounding mathematics. The attribute suggests the length of the line—experiment a bit, since it is not super-precise.

6.8.9 Page Breaks for Tall Display Mathematics

For print output, do nothing additional and LaTeX will do its best to break your display between lines. You can turn this behavior off by setting the `@break` attribute on the `<md>` or `<mdn>` to the value `no`. Once you do this, you can then selectively allow a page break after a given `<mrow>` by setting the `@break` attribute on the `<mrow>` to the value `yes`.

6.8.10 Your Macros

These go in the `<docinfo>` section, wrapped in a `<macros>` element. Keep them simple—one or two arguments, and one-line definitions. This is not the place to be fancy, and not the place to try to end-run the structural aspects of PreTeXt. The idea is to define something like `\adjoint{A}` for the matrix `A` to be a superscript asterisk, and later you can change your mind and use a superscript dagger instead. Keep in the spirit of PreTeXt and use readable, semantic macros. For example, do not use `\a{A}` for the adjoint of `A`.

PreTeXt will use your macros correctly for print and for HTML, after erasing whitespace from the left margin, and stripping LaTeX comments.

6.8.11 Punctuation After Display Math

If a chunk of displayed math concludes a sentence, then the sentence-ending punctuation should appear at the conclusion of the display. (And certainly not at the start of the first line after the display!) But do not author the punctuation within the mathematics element, put it afterwards, where it logically belongs.

More specifically, place a sentence-ending period (say) *immediately* after the closing of an `<me>`, `<men>`, `<md>`, or `<mdn>` element. PreTeXt will place the period in your output in the right place and in the right way. (By using LaTeX's `\text{}` macro, if you are curious to know the details.) Here is an example. The XML source

```
<md>
    <mrow>(a+b)^2</mrow>
</md>.  Now...
```

will render as

$$(a+b)^2.$$

Now...

This all applies more generally to clause-ending punctuation, such as a comma. Take notice of the requirement that the punctuation must be *immediately* after the closing tag of the math element, otherwise it will not migrate properly. For example, do not interrupt the flow with whitespace, or an XML comment, or anything else.

Here is a technical subtlety that will demonstrate some of the inner machinery of PreTeXt and our conversions. In your work, locate a theorem that has some numbered display mathematics (`mdn`) which is at the end of a sentence, and which you have authored as described above. In HTML output, test a cross-reference (`xref`) to the theorem and you will see the period for the end of the sentence at the end of the display, where it should be. Now test a cross-reference (`xref`) to one of the numbered equations. First, the knowl will contain the entire display, to provide context, but it also will not contain the period, since the rest of the sentence is not in the knowl and so the period is not necessary.

6.8.12 Additional Packages

Generally, you cannot add additional packages for use within mathematics. The exception is a package with support available optionally within MathJax. And it must have the same name as its normal LaTeX version. Then set a `docinfo/latex-preamble/package` element to be the common name of the package. (The `cancel` package is one such example.)

Then the supported macros of the package will be available with your mathematics elements, and you can use them within other macro definitions. We do not guarantee the absence of conflicts with other packages in use, even if employed by PreTeXt. Nor do we support debugging such conflicts.

6.8.13 Extras

There are two existing additional options, which we might want to remove some day for technical reasons. Macros from the `extpfeil` extensible arrows package are availble by default, and an `\sfrac{}{}` macro is available for appealing inline "slanted fractions."

6.9 Lists

A **list** is an unusual construction, even if everybody knows exactly what one is. We view the list itself as a containier of various chunks of text, while those chunks of text are the **list items**. Each item has a **label** to identify it (which is quite different from how LaTeX uses the term "label").

Markup and processing is complicated by the possibility of a list item containing a list, resulting in **nested lists**. We simplify this problem by *requiring* that a list appear within a paragraph (`<p>`), see Subsection 6.1.3.

One of the three exceptions is the possibility to put a list into a block that earns a caption and a number, using the `<list>` element, see Subsection 6.9.6.

The final subsection contains some examples that you may wish to consult as you read this section.

6.9.1 Ordered, Unordered, Description Lists

An **ordered list** has items with labels that are naturally ordered (usually numerically or alphabetically). We borrow from HTML, and use the `<ol>` tag to construct an ordered list. Some commentators suggest an ordered list should only be used when the order of the content is important. So the steps in a recipe would belong in an ordered list, but the shopping list when you go to the store need not be an ordered list.

An **unordered list** has items with labels that have no inherent order and so are usually symbols like circles, disks, squared, etc. We borrow from HTML, and use the `<ul>` tag to construct an unordered list.

A **description** list has items that have short pieces of text as their labels. We borrow from HTML, and use the `<dl>` tag to construct a description list.

Ordered lists are used as part of `<objectives>` ((((topic-objectives)))) and exercises (Section 6.10). Any of the three lists may occur inside the `<list>` element (below, Subsection 6.9.6). Otherwise, a list must occur within a paragraph, `<p>`. This means that to place a list within a list item of another list, the list item must contain a paragraph.

6.9.2 List Labels

Do nothing, and your ordered and unordered lists will get sensible default labels. They are consistent in the following sense. If your list has two items, and each of the two items contains a list, then these two lists will use the same type of label.

For a description list, you author each label as part of each list item, as discussed below in the discussion of list items.

If you want to change how an ordered list is labeled, then you use the `@label` attribute on the `<ol>`, whose value is a **format code**. This string contains one of five codes (a single character), which may be surrounded by other characters, excluding the five codes. For example `label="(A)"` will produce uppercase letters wrapped in parentheses: (A), (B), (C), ... The extra formatting works well in a conversion to LaTeX, but is not possible technically in a conversion to HTML, so it should be considered decorative, and not relied upon for meaning. The formating does not carry through to the numbers of list items in cross-references.

If you want to change how an unordered list is labeled, then you use the `@label` attribute on the `<ul>`, whose value is a **format code**. This string contains one of three codes in the table below. Then every item of the list will have that symbol as its label.

Code	Realization
1	Arabic numerals
a	Lowercase letters
i	Lowercase Roman numerals
A	Uppercase letters
I	Uppercase Roman numerals

Table 6.9.1: Ordered List Labels

Code	Realization
disc	Filled small circle, aka a bullet
circle	Small circle
square	A square

Table 6.9.2: Unordered List Labels

Default label types are assigned to each level of nested lists in the order shown in the table, and cycle back to the top of the table if necessary.

Start with the defaults, and experiment as needed. See the examples below for some extreme (and unwise) customizations of labels.

For a description list, possible labels are more varied than what you can express with an XML attribute. So the list item must have a `<title>` element (see below). This should be short text, and may contain inline mathematics. It is often rendered in bold, so be aware that some markup may get lost. Perhaps for obvious reasons, do not include footnotes, cross-references, or display mathematics. The `<dl>` element has a `@width` attribute, with possible values `narrow`, `medium`, and `wide`. The default is `narrow`. This is a *hint* about how

much text you have in these labels, and in certain presentations may make better use of horizontal space on a page.

6.9.3 List Items

So now you have a list all organized as a container. What do you put in it? List items, using the `<li>` tag, again borrowed from HTML, and independent of the type of list.

A list item could be really simple, maybe just one or two words. Then you can use, and conceptualize, an `<li>` element as not much different from a `<p>` element, and the rules about content are not much different. Even several full sentences, with some intermediate displayed mathematics, is fine.

But once you want two paragraphs in a list item, then you need to structure the contents of the item. So a list item might have five paragraphs in it, requiring five `<p>` elements. Notice that this is how you nest lists. Make a list item, include a paragraph, then put the subsidiary list into the paragraph. Indeed, this is the only way to nest lists. A consequence of this is that the only way to have an unstructured list item is if it is a terminal item, like the leaf of a tree.

Other items may be interspersed among the paragraphs of a list item, such as a chunk of verbatim text delimited by a `<pre>` tag. But anything with a number, such as a `<figure>` or `<remark>` is banned, in part because the consequences for numbering and organization become too complicated. Imagine a remark, and a paragraph of the remark has a list. Fine so far. But if the items of that list can again contain remarks, the possibilities become endless. You can use a `<sidebyside>` in a structured list item, which can in turn hold an `<image>`, `<tabular>`, or similar. But you cannot place items in a such a `<sidebyside>` that are numbered, so a `<figure>` or `<table>` is not possible. A general rule is no numbered components in a list item. Computational components, such as `<sage>` are also banned from list items due to the difficulty of converting them into electronic computational notebooks with a relatively flat structure.

A list item of a description list must have a `<title>` element, to provide the text of the label. Now that the list item has some structure, the remainder must also be structured, typically with some paragraphs, as discussed above. In other words, the earlier option of employing an `<li>` element just like a `<p>` element is not available in a description list. Further, given the complexity of presenting a description list, it can *only be a top-level list*. It can contain the two other types nested within its list items.

6.9.4 Cross-References to List Content

Note that a list is a container, so it cannot be the target of a cross-reference, and so the three types of lists cannot have an `@xml:id` attribute. But you may well be able to point at some other structure (e.g., a `<remark>`) with a paragraph containing a list of interest. If this seems overly restrictive, read below about named lists.

By contrast, a list item, `<li>`, is not a container, and does contain content. Further, a list item of an ordered list has a label that is natural text for a cross-reference. So in this situation, the list item can have an `@xml:id` attribute. But note that the "number" of a list item of an ordered list, which is nested inside a list item of an unordered list, is not defined, so a cross-reference by number can fail.

The "number" of a list item, mostly for the purposes of a cross-reference, is the concatenation of all of the individual labels in the containing lists, outermost first. For example, from the example lists below, the list item with content "Walleye" has number 2.I. These are indivisible, there is no way to get a component, excepting leading subsequences obtained by using an `@xml:id` on a containing list item. Note that the format codes never become part of the number.

6.9.5 Lists in Columns

You can control the number of columns used to layout an ordered or unordered list (but not a description list). On the `<ol>` or `<ul>` use a `@cols` attribute with values `2` through `6`. (`1` is the default.)

We do not yet (2018-03-28) have enough technical confidence to allow an author to specify a row-major order versus a column-major order for the layout. So understand that this is can be an implementation choice for a particular conversion, and can vary across implementations. If this is critical to conveying *meaning*, and not an aesthetic preference, then maybe consider using a `<table>` or `<tabular>` (Section 6.12).

Best Practice 6.9.3 Use Only a Few Columns for Lists. Anything more than three columns tends to get very crowded horizontally. Think twice about using more than that, and realize that six columns should be a ridiculously generous upper limit, and not a promise of good behavior in final output.

6.9.6 Named Lists

As mentioned above, it is not possible to have a list be the target of a cross-reference. Should an entire list be *so important* that you need to point to it from elsewhere, then make it a **named list** by wrapping it in the `<list>` tag.

This element can begin with an optional `<introduction>`, then has a single, required list, which may be any of the three types. It concludes with an optional `<conclusion>`. It can have an `@xml:id` attribute, which in a way is the whole *raison d'être* for this construction. It will be numbered when rendered, and so also requires a `<caption>`. You may give it an optional `<title>`, even if implementation is spotty presently (2018-03-28). (You did say it was important, no?) You might think of it as a specialized `<figure>`, it is entirely similar.

Since this element associates a number, caption, and title, to an entire list, we call it a "named list". What should we call a list that is authored within a paragraph and cannot be the target of a cross-reference? We call it an **anonymous list** when we want to make the distinction.

6.9.7 Exceptional Lists

We use the tags for lists in a few situations outside of anonymous lists inside paragraphs and named lists. These include the items within an objectives, subparts of an exercise, and within panels of a side-by-side. See those topics to learn about subtle differences in use.

6.9.8 Examples of Lists

To illustrate this section, we offer three too-elaborate examples. Take these as compact examples of what is possible, and not best practice in your writing. We also use these to illustrate cross-references to list items, see Subsection 6.6.6.

We have a paragraph that begins with anonymous list of species that live in water (maybe partially), which necessarily is placed inside a paragraph. The roman numerals purposely do not have any extra adornment in the LaTeX version (but may for HTML output).

1) Amphibians
    a. Frog
    b. Salamander
    c. Newt
    d. Toad
2) Freshwater Sport Fish
    I Walleye
    II Bass
    III Trout
3) Saltwater Sport Fish
    (A) Salmon
    (B) Halibut
    (C) Marlin

Within the same paragraph, we transition to an unordered, two-column, list of some germs:

- Bacteria
  - *Staphylococci*
  - *Streptococci*
  - *Salmonella*
- Viruses
  - *Varicellovirus*
  - *Orthopoxvirus*

This sentence concludes our (small) paragraph on small and large organisms.
A named list, only to test cross-references.

A: i. A and i
ii. A and ii

B: (a) B and a
(b) B and b
(c) B and c

C: <I> C and I
<II> C and II
<III> C and III

List 6.9.4: A two-deep ordered list

An example of a description list, anonymously in a paragraph.

Red The color of the sun at sunset.

Blue The color of a clear sky.

Aqua The color of shallow tropical waters.

$x^2 + y^2$ Definitely not a color

Remark 6.9.5 Best Practice. Lists are a very attractive device. Hopefully the discussion above has convinced you that they are more complicated than they first appear. Think carefully before using one, and consider if some other structure (`<paragraphs>`, `<sidebyside>`, a subdivision) might do a better job of organizing and communicating your meaning. And if a list is really necessary, consider if it should be named or anonymous, heavily-nested or nearly-flat, with columns, or with long or short content in the items. Cross-references *from* the items of a list *to* more complicated structures is another device that works well.

6.10 Exercises and their Solutions

As described in Section 3.8 an `<exercise>` can be placed in many different locations, and a `<project>` has similar features. It is critical to understand that you want to author any hints, answers, or solutions immediately following the statement of an exercise. If your PreTeXt source is public, and you would like to keep some aspects of the solutions private, then read the *Publisher's Guide* for some practical advice. See the *Publisher's Guide* as well for information on creating a standalone *Solutions Guide*. We concentrate here on techniques for controlling visibility and location of the components of exercises within your primary output.

6.10.1 Exercises, Original Versions

A <hint> to an <exercise> renders nicely in a knowl, right below the exercise statement, as part of a conversion to HTML. For a conversion to LaTeX/PDF/print, you might wish to display a hint, visibly, as part of the exercise, or you may wish to park the hint in a *Hints to Exercises* division in the back matter. To control visibility of the components of exercises (and projects) there are fifteen switches you can use as string parameters on the command line, or in an extra XSL file. Each takes values of yes or no.

exercise.inline.hint	exercise.inline.answer
exercise.inline.solution	
exercise.divisional.hint	exercise.divisional.answer
exercise.divisional.solution	
exercise.worksheet.hint	exercise.worksheet.answer
exercise.worksheet.solution	
exercise.reading.hint	exercise.reading.answer
exercise.reading.solution	
project.hint	project.answer
project.solution	

Table 6.10.1: Exercise Component Switches, Original Versions

6.10.2 Exercises, Solutions Versions

Exercises, and their components may be duplicated easily, to provide a back matter appendix with solutions, or within each division. For example, you can easily create an end-of-chapter division with solutions to every inline exercise throughout the chapter and solutions to all the divisional exercises from each section of the chapter.

The <solutions> element will create an entire division, semi-automatically. You can provide a <title>, an <introduction>, and <conclusion>. The remaining content is statements, hints, answers, and solutions to exercises (and projects).

If <solutions> is a child of <backmatter>, then an appendix will be generated, and covering <exercise> from the entire <book> or <article>. If <solutions> is a child of a division, then a new subdivision is created and the scope is all <exercise> for the division. So, for example, a <solutions> placed inside a <chapter> will render as a division that looks like a <section> and will include components of all the exercises (at any level) contained within the <chapter>.

An author filters the types of exercises, and their components, through attributes of the <soutions> element. For example

```
reading="hint answer"
```

would cause every <exercise> within each <reading-questions> to have its <hint> and <answer> displayed, but not its <statement> nor its <solution>. These are the attribute names and the possible values.

inline	statement
divisional	hint
reading	answer
worksheet	solution
project	

Table 6.10.2: Attributes (left) and Values (right) for <solutions> element

So, PreTeXt source like

```
<section>
  <title>Tropical Bird of Paradise<title>
  ...
  <solutions worksheet="hint solution" project="hint solution">
    <title>Hints and Solutions to Worksheets and Projects<title>
  </solutions>
</section>
```

would generate an entire subsection with hints and solutions to every worksheet and every project, located anywhere (including in subsections and subsubsections) in the section on Birds of Paradise.

6.11 Images

6.11.1 Raster Images

A **raster image** is an image described pixel-by-pixel, with different colors and intensities. Photographs are good examples. Common formats are Portable Network Graphics (PNG) and Joint Photographic Experts Group (JPEG, JPG), which will both work with `pdflatex` and modern browsers. JPEG are a good choice for photographics since they are compressed on the assumption they will be viewed by a human, while PNG is a lossless format and good for line art, diagrams and similar images (if you do not have vector graphics versions, see below).

To use these images, you simply provide the filename, with a relative path. A subdirectory such as `images` is a good choice for a place to put them. It is your responsibility to place these images where the LaTeX output will find them or where the HTML output will find them. The XML would look like:

```
<image source="images/crocodiles.png" width="50%" />
```

Typically you would wrap this in a `<figure>` that might have an `@xml:id` attribute for cross-references, with or without a caption. There is no `@height` attribute, so the aspect ratio of your image is your responsibility outside of PreTeXt. The `@width` attribute is a percentage of the available width of the text (outside of a `<sidebyside>` panel). Default width is typically 90%.

You may also provide a `<description>` which will aid accessibility for electronic formats. Keep such readers in mind and provide as much description as possible. Keep the markup simple, since this will typically migrate to an HTML attribute that cannot contain any structure. Be careful to avoid double-quotes. For example,

```
<image source="images/crocodiles.jpeg" width="50%">
    <description>Five crocodiles partially submerged in the shallows.</description>
</image>
```

6.11.2 Vector Graphics

An image is a **vector graphic** if the file describes the geometric shapes that constitute the image. So a simple diagram would be a good candidate, but a photograph would not. Popular formats are Portable Document Format (PDF) and Scalable Vector Graphics (SVG). You will get the best results with PDF images in LaTeX output and SVG images for HTML. The principal advantage of these formats is that they scale (big or small) smoothly, along with fonts. This is critical when you cannot predict the screen size for a reader of an electronic version.

Unless you describe these images with a language (see next subsection), you are responsible for providing the PDF and SVG versions. The `pdf2svg` utility is very useful if you have PDF images only. To use these images, you simply follow the instructions above, but do not include a file extension. This alerts the conversion to use the best possible choice. So presuming we had files `images/toad-life-cycle.pdf` and `images/toad-life-cycle.svg`, an example would be:

```
<image source="images/toad-life-cycle" width="85%">
    <description>Diagram of the four stages of a toad's life.</description>
</image>
```

6.11.3 (*) Images Described by Source Code

To be written once elements and tags solidfy, see sample article for examples.

6.11.4 Image Archives

As an instructor, you might want to recycle images from a text for a classroom presentation, a project handout, or an examination question. As an author, you can elect to make images files available through links in the HTML version, and it is easy and flexible to produce those links automatically.

First, it is your responsibility to manufacture the files. For making different formats, the `mbx` script can sometimes help (Chapter 9). The Image Magick `convert` command is a quick way to make raster images in different formats, while the `pdf2svg` executable is good for converting vector graphics PDFs into SVGs. Also, to make this easy to specify, different versions of the same image must have identical paths and names, other than the suffixes. Finally, the case and spelling of the suffix in your MBX source must match the filename (e.g. `jpg` versus `JPEG`). OK, those are the ground rules.

For links for a single image, add the `@archive` attribute to the `<image>` element, such as

```
<image  ...  archive="pdf svg">
```

to get two links for a single image.

To have every single image receive an identical collection of links, in `docinfo/images` place an `<archive>` element whose content is the space-separated list of suffixes/formats.

```
<archive>png JPEG tex ods</archive>
```

will provide four links on every image, including a link to an OpenDocument spreadsheet.

For a collection of images that is contained within some portion of your document, you can place an `@xml:id` on the enclosing element and then in `docinfo/images` place

```
<archive from="the-xml-id-on-the-portion">svg png</archive>
```

to get two links on every image *only* in that portion (chapter, subsection, side-by-side, etc.). The `@from` attribute is meant to suggest the root of a subtree of your hierarchical document. If you use this, then *do not* use the global form that does not have `@from`.

You may accumulate several of the above semi-global semi-local forms in succession. An image will receive links according to the last `<archive>` whose `@from` subtree contains the image. So the strategy is to place general, large subtree, specifications early, and use refined, smaller subtree specifications later. For example,

```
<archive from="the-xml:id-on-a-chapter">svg png</archive>
<archive from="the-xml:id-on-the-introduction">jpeg</archive>
<archive from="the-xml:id-on-a-section-within" />
```

will put two links on every image of a chapter, but just one link on images in the introduction, and no links at all on every image image within one specific section. Again, do not mix with the global form. You can use the root document node (e.g. `<book>`) for `@from` to obtain a global treatment, but it is unnecessary (and inefficient) to provide empty content for the root node as first in the list—the same effect is the default behavior.

Notice that this facility does not restrict you to providing files of the same image, or even images at all. You could choose to make data files available for each data plot you provide, as spreadsheets, or text files, or whatever you have, or whatever you think your readers need.

Finally, "archive" may be a bit of a misnomer, since there is no historical aspect to any of this. Maybe "repository" would be more accurate. Though for a history textbook, it might be a perfect name.

6.12 (*) Tables and Tabulars

This section needs much more work.

Note that tables may be constructed using the LaTeX Complex Table Editor tool online at `latex-tables.com`and then exported in PreTeXt syntax.

6.13 (*) Program Listings

6.14 Side-by-Side Panels

Documents, pages, and screens tend to run vertically from top to bottom. But sometimes you want to control elements laid out horizontally. A <sidebyside> is designed to play this role. It is best thought of as a container, enclosing **panels**, and specifying their layout. Examples include three images, all the same size and equally spaced. Or a poem occupying two-thirds of the available width, with commentary adjacent in the remaining third. Or an image next to a table. But the most common use may be a single image (with no caption, and hence no number), whose width and horizontal placement are controlled by the layout.

See the schema for the exact items that are allowed in a <sidebyside>. To author, just place these items within <sidebyside> in the order they should appear, left to right. Then you add attributes to the <sidebyside> element to affect placement.

Instead of placing a @width attribute on each item, instead place this on the <sidebyside> element. A single @width will use the same value for each panel. For different widths, use the plural form @widths and provide a space-separated list of percentages. The default is to give each panel the same width, and as large as possible, which will result in no gap between panels.

The margins can be specified with the @margins attribute, which if given as a single percentage will be used for both the left and right sides. You may also specify asymmetric left and right margins with two percentages, separated by a space, in the same attribute. An additional option is to use the value auto which will set each margin to half of the (common) space between panels. This is also the default. In the case of a single panel, the left margin, right margin, and panel width should all add up to 100%.

Once the widths and margins are known, any additional available width is used to create a common distance separating panels. (Which is not possible when there is just a single panel.)

Independent of horizontal positioning, individual panels may be aligned vertically. The attribute is @valigns and its value is a space-separated list of top, middle, and bottom. The singular version, @valign, is used to give every panel the same alignment, using the same keywords. The default is to have every panel at the top.

We could give lots of examples, but instead it might be best to just experiment. Error-checking is very robust, so it is hard to get it too wrong. OK, we will do just one to help explain. Suppose a <sidebyside> contains three panels and has layout parameters given by

```
<sidebyside widths="20% 40% 25%" margins="auto" valign="middle">
```

Then there will be 15% of the width left to space out the panels. The two gaps are each 5% of the width, and the remaining 5% is split between the margins at 2.5% each. And the vertical midlines of each panel are all aligned.

For a single panel with no attributes, the panel will occupy 100% of the width. A single panel with a specified width will get equal (auto) margins, resulting in a centered panel.

Captioned items as panels deserve special mention. These will continue to be numbered consecutively, with one exception. If you place a <sidebyside> inside of a <figure>, then the <figure> will be numbered, and the captioned items inside the <sidebyside> will be **sub-captioned**. In other words, the second captioned panel of a <sidebyside> inside Figure 5.2 would be referenced as Figure 5.2.b.

An <sbsgroup> ("side-by-side group") contains only <sidebyside>, which are displayed in order. However, all of the layout parameters allowed on a <sidebyside> may be used on an <sbsgroup>. This might allow a collection of fifteen images to be laid out in three rows of five images each, with widths and spacing identical for each row because the parameters are specified on the <sbsgroup> element. In this way, simple grids can be constructed. Note that any layout parameters given on an enclosed <sidebyside> will take priority over those given on the <sbsgroup>. Captioning behavior extends to an entire <sbsgroup>.

Since <sidebyside> and <sbsgroup> are containers they cannot be referenced and so do not have an @xml:id. However, you can reference their individual contents if they are captioned, and you can reference an enclosing <figure>.

Generally, a <sidebyside> or <sbsgroup> can be placed as a child of a division, or within various blocks, such as <proof> for example. See the schema for (evolving) specifics.

It should be clear now that a <sidebyside> is more about presentation than most PreTeXt elements,

though there is some semantic information being conveyed by grouping the panels with one another.

6.15 (*) Front and Back Matter

6.16 (*) Index

TODO

6.17 (*) Notation

TODO

6.18 (*) Automatic Lists

6.19 URLs and External References

6.19.1 URLs to External Web Pages

The `<url>` tag can be used to point to external items (as distinct from other internal portions of your current document, which is accomplished with the `<xref/>` element, Section 3.3). It always needs a value for the `@href` attribute, most likely a URL. Most of this time, this will point to some resource available on the Internet but it could be a file on the system hosting your document, perhaps using a relative address (but see the rest of this section for some cautions). Here are three scenarios:

- The `<url>` element is empty. Then the value of the `@href` is also used for the visible text of the link, verbatim, and usually in a monospace font. Use **percent-encoding** (aka **URL encoding**) for the `@href` attribute to include special characters, such as spaces.
- The `<url>` element has content. Now the content should be authored as you would any other text in a sentence. Potentially problematic characters, such as a tilde should be authored with provided empty elements (as distinct from being authored literally in the `@href` attribute).
- The `<url>` element has content, but you want this content to look like a URL. Use the `<c>` element around the content, and follow the rules for verbatim text. I do this often for simple URLs that point to the top level of a website. The `@href` is a complete URL like `https://pretextbook.org/` but for content I use a less-imposing reader-friendly version like `pretextbook.org`.

You can place an external reference into a `<title>` element, but conversions may choose to simply render it as text, and not as an active link.

For LaTeX output it gets quite tricky to handle all the various meanings of certain escape characters in URLs in more complicated contexts (such as tables, footnotes, and titles), so there may be some special cases where the formatting is off or you get an error when compiling your LaTeX. We handle most of these situations, but we always appreciate reports of missed cases.

6.19.2 Characters in URLs

A URL can have a **query string**, which has a list of parameters following a question-mark. The parameters are separated by ampersands (&), which will need to be escaped, so as to not confuse the XML processor. So use `&amp;` anywhere the ampersand *character* is necessary, such as a `@source` attribute, or a monospace version of a URL achieved with a `<c>` element. Also, the question-mark character should *not* be URL-encoded (`%3F`) (despite advice just given above), so if necessary edit it to be the actual character. General advice about special characters in XML source can be found in Section 3.13.

6.19.3 URLs to External Data Files

The <url> element can be used to make data files available to your reader. Consider the example of a spreadsheet containing a large data set that a reader needs to analyze as part of an exercise. Here are our recommendations on how to accomplish this:

- If the file is hosted on some server unassociated with your project, and does not have a license compatible with your project, then just set the @href to the complete address. Be sure to include enough of the address for the reader of a a print version to be able to type in the URL, either as the content of the <url> or in close vicinity.
- If you authored the spreadsheet, or you are allowed to legally copy and distribute it, then place it on your server where you host your book project. Then do as above and use the full URL for the @href attribute, with a visible version available for PDF and print versions.
- If you have control over the placement of the file, you can host it on your server, and use a URL relative to the location of your HTML, PDF, or other files that comprise your document. This might be a good choice if your book will be posted many places and you can give it to others as an archive, like a *.zip file. It is a bad idea if a reader downloads a PDF without the data file following along and remaining in the same relative location. It is an impossible idea if your document gets printed on paper and there is no idea what a relative URL means and there is not even a link to click on.

Consider your audience and think about how much guidance they need about using context menus or helper/viewer applications to make use of the file formats you are providing. This advice may be different depending on the type of files and the types of output for your document.

6.20 Video

A video is a natural way to enhance a document when rendered in an electronic format, such as HTML web pages. It might be additional information that is hard to communicate with text (marine invertebrates swimming), a lecture or presentation that augments your text, or even some artistic work, such as a symphony legally hosted on YouTube, when you could never hope to get copyright clearance yourself.

PreTeXt supports videos you own and distribute with your source, videos shared openly on the Internet via stable URLs, and videos available on YouTube. Go straight to to the end of this section to see how easy it is to incorporate a YouTube video.

HTML5 web browsers are able to play video files in threee formats, summarized in the following table.

Format	Extension	Reference
Ogg, Theora	.ogg	Free and open, Wikipedia
WebM	.webm	Royalty-free, Wikipedia
MPEG-4	.mp4	Patent encumbered, Wikipedia

Table 6.20.1: HTML5 video formats

6.20.1 Video Element

The <video> element is used to embed a video in output formed from HTML. Subsections below describe the different ways to indicate the source of the video. The video may be placed inside a <figure> or can be a panel of a <sidebyside>. The former will have a caption, be numbered, and hence can be the target of a cross-reference (<xref>). The latter is anonymous, but allows for horizontal layout, and combinations with other panels.

Size is controlled by a @width attribute expressed as a percentage (on the <video> element when used in a figure, or as part of the <sidebyside> layout parameters). Height is controlled by giving the **aspect ratio** with the @aspect attribute on the <video> element. The value can be a ratio expressed like 4:3 or a

decimal number computed from the width divided by the height, such as `1.333`. The default for videos is a 16:9 aspect ratio, which is very common, so you may not need to specify this attribute.

Options include specifying a `@start` and an `@end` in seconds as integers (no units) if you only want to highlight a key portion of a video. The `@play-at` attribute can take the following values

`embed` Play in place (the default action).

`popout` Play in new window or tab, at 150% width.

`select` Provide the reader the choice of the other two options.

In an educational setting, sometimes the preview images provided by YouTube can be distracting, or for an author-provided video you may wish to provide your own preview image. The `@preview` attribute can take on the following values

`generic` PreTeXt supplies a Play-button image.

`default` Whatever the video playback provides. This is identical to simply not including `@attribute` at all

Path to an image file Typically, this will be a relative path, starting with `images/`. This image will be used as preview for the online version and the print version.

6.20.2 Author-Provided Videos

If you own and possess your video content, then you can distribute it with your PreTeXt source, and it can be hosted as part of your HTML output. Then the `@source` should be a relative file name that points to the file containing the video. If you are able to provide more than one of the three formats in Table 6.20.1, then you can provide the filename *without an extension*. If a browser cannot play one format, it may be able to play another. PreTeXt will write the code to make that happen, preferentially in the order of the table (more open formats first!). In other words, you can provide files in more than one format and increase the likelihood that a reader's browser will find a format it can playback.

6.20.3 Network-Hosted Videos

If a video is shared openly on the Internet, you can simply provide the full URL for the `@source` attribute. All the other attributes are the same as for the author-provided case, above. Read Subsection 6.19.2for some considerations when authoring a URL, since there are a few gotchas.

You can frequently discover the URL of a video by first playing it, and then using a context menu (e.g. via a right-click) to reveal an option to copy the video's location. However, note that there are various techniques sites use to make such a URL temporary, or otherwise unusable. So do some research about potential uses and test carefully. Our example below is provided from a United States government site.

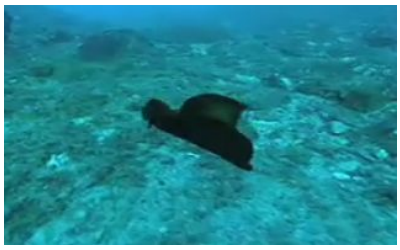


Figure 6.20.2: Sea Hares, Flower Garden Banks National Marine Sanctuary

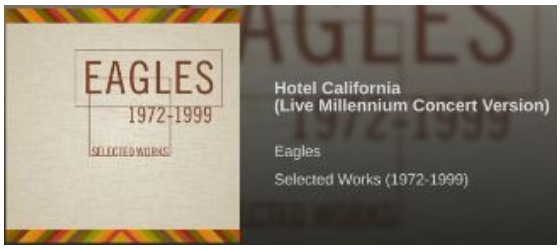
6.20.4 YouTube Videos

For a video hosted at YouTube, find the 11-character identification string in the address of a video you are viewing. It will look something like `hAzdgU_kpGo`. Then, instead of the `@source` attribute, simply provide this identification string as the `@youtube` attribute, such as

```
youtube="hAzdgU_kpGo"
```

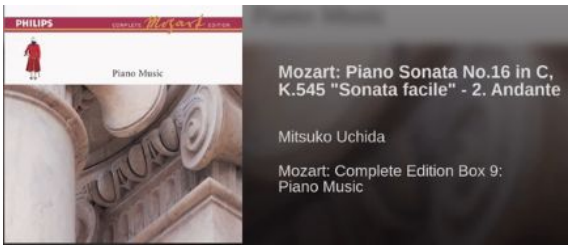
That's it. All of the options above are then implemented and realized with YouTube's embedded player.

This can be a great way to incorporate popular or artistic content, legally, which might be difficult or costly to acquire through copyright clearance.



YouTube: `https://www.youtube.com/watch?v=YW4dzWE1SUs`

Figure 6.20.3: The Eagles, "Hotel California"



YouTube: `https://www.youtube.com/watch?v=1Yv_AINhVn4`

Figure 6.20.4: Mozart, Piano Sonata in C Major, K. 545, II

The `mbx` script (Chapter 9) may be used to download the provided preview images for YouTube videos (only). Filenames will be formed from the `@xml:id` of the `<video>` element. These will be used in static versions of output, such as print. Once custom preview images are implemented for author-hosted video, their static representation will improve.

Additionally, a YouTube **playlist** can be included in one of two ways. You may set the `@youtube` attribute to be a space-separated list of several video IDs (an "itemized" playlist). Alternatively, you may set the `@youtubeplaylist` attribute to a YouTube playlist ID (a "named" playlist). At present, a named playlist will not get a thumbnail image from Youtube, and either the "generic" thumbnail will be used or you can supply your own `@preview`.

YouTube: https://www.youtube.com/playlist?list=PLGRecdYLYDNdX5NEkS3KH-ZtCqc2f8lJ_

Figure 6.20.5: YouTube Playlist

Make a feature-request if a scheme similar to the one for YouTube, but for some other video-hosting service, would be useful for your project.

6.21 (*) Music

TODO: Scholarly works discussing music may use notes and chords in text, and displays of sheet music are easily supported. (TODO: add some discussion to Chapter 3.)

6.22 (*) Units of Measure

6.23 Unicode Characters

PreTeXt supports (and encourages) the use of Unicode characters. Here are some relevant comments.

- Unicode characters will migrate well to any output format based on HTML. Most browsers will have a variety of fonts with glyphs to realize these characters.
- LaTeX will not always behave as smoothly. For openers, you definitely will want to use the `xelatex` engine to build a PDF. Then you need to be sure your system has a font with the necessary characters and you make the font known to `xelatex`. We are working out the details of the best way to accomplish this.
- How do you get a Unicode character into your source? In part this is specific to your operating system and editor, so is outside the scope of this guide, but we have hints below for popular operating systems.
- You can always place a Unicode character in your source using XML syntax. The first thing an XML parser will do is convert this syntax into a character. The number of the `SECTION SIGN` in hexadecimal is `A7`, so the syntax `&#xA7;` is identical to the character §. Of course, this will get tedious fast.
- The Full Unicode Input utility at `www.cs.tut.fi/~jkorpela/fui.html8` will allow you to specify a chunk of 256 consecutive Unicode numbers and then you can click on characters to make a string of several or many. You can cut/paste these into your source, or convert the whole lot to XML syntax all at once.
- Unicode characters have standardized names. You can find these, and more information, including font support, at the Unicode section of FileFormat.info, `www.fileformat.info/info/unicode/`. If you are struggling to find a specific character, then using this site's name in a search will often quickly locate what you need. Be sure to experiment with the test pages there for browser and font support (including checking your local configuration).
- ***Warning***: do not use Unicode characters as a way to get mathematical symbols (that is delegated to our use of LaTeX syntax). And do not use Unicode when we have provided an empty element for a

character, especially when that character may be used in a markup syntax for some output, such as LaTeX, HTML, JSON, Markdown, ...

For example, if you put many naked hash symbols (#) in your source, then you will get nice HTML, but when you try to get print from a PDF from LaTeX you will have a train wreck on your hands when you compile the LaTeX. Instead, be sure to always use the provided `<hash />` element. *Always*. Other empty elements are conveniences, which spare you from looking up Unicode numbers and make your source more readable, rather than a necessity to avoid special characters. An example is `<times />`, for use outside of a strictly mathematical setting: "I bought a 2×4 at the lumberyard."

6.23.1 Unicode Support in OSX

Mitch Keller reports on 2017-01-12 a way to get some popular characters with OSX. Use the Keyboard preference pane under System Preferences. In there, you can enable

`Show Keyboard, Emoji, & Symbols Viewers in menu bar`

Once you activate the keyboard viewer, you get a keyboard on your screen. When you hold down `opt`, it shows you what other symbol you would get if you push `opt+letter`. For instance, `opt+w` gives an upper-case Greek sigma and `opt+=` gives a not-equals sign (neither of which we can handle when processing the latex version of this guide). To get ä, you type `opt+u` and then hit `a`. This is illustrated by the keys for diacritical marks being highlighted in orange while holding opt. The shift key can have an effect to produce variations of some characters, such as quote marks (dumb versus smart).

6.23.2 (*) Unicode Support in Linux

6.23.3 (*) Unicode Support in Windows

6.24 (*) Testing Sage Examples

6.25 Xinclude Modularization

The `xinclude` mechanism is not part of PreTeXt, *per se*. It is of some use for organizing your authoring, so you do not have mammoth files open in your text editor. As discussed in Section 4.2 there is very little value in modularizing so much that you have many very small files, and also almost no benefit whatsoever to using directory structure to duplicate the inherent tree-like structure of XML. Many small files, or deeply-nested directories, seem to be of little help and can cause more headaches than they are worth.

The `xinclude` mechanism automatically introduces a `@xml:base` attribute, which we need to account for in the RELAX-NG schema (Chapter 5). So we limit which PreTeXt elements may be the root element of an included file. The rough, general rule is that if an element *can* have a title, then it can be the root element of an included file. So in particular each of the divisions (`<chapter>`, `<section>`, etc.) is a candidate.

One special exception to this restriction is the use of text files, containing absolutely no markup at all. Two good examples are the `<input>` child of a `<program>` or the `<latex-image-code>` element used to describe an `<image>` by source code that LaTeX understands.

In both cases you can put the text content of these elements in a separate file, use the `@href` attribute of `<xi:include>` to point to the file, and then the twist is to set the `@parse` attribute to the value `text`. This has two general benefits. First, you now *cannot* have any XML in the file, so you do not have to have a single root element for the file (and so the schema imposes no restrictions). Second, you do not need to escape any problematic characters like ampersands and angle brackets (Section 6.5), nor use the misunderstood `CDATA` mechanism.

Additionally, in the case of `<latex-image-code>` you can park unsightly code away in files so you do not have to look at it, or you can create a small driver LaTeX program to test each one, or even better, you may

want to use the same image more than once (maybe in different figures?) and can just include it repeatedly, while only ever editing the single copy.

Finally, the <input> and <output> children of <program>, <sage>, and <console> are also candidates for this device. In particular, you may want to have the code for a program in its own file where you can test it easily with an interpreter or compiler. There is one gotcha. If you were to put a newline between <input> and <xi:include> there is the very real potential of unwanted whitespace bleeding into your PreTeXt output. Our suggested remedy uses an example from Bob Plantz. Convert

```
<program language="c">
    <input>
        <xi:include parse="text" href="intAndFloat.c"/>
    </input>
</program>
```

to

```
<program language="c">
    <input><xi:include parse="text" href="intAndFloat.c"/></input>
</program>
```

There are some fancy XSLT tricks you can employ to use more complicated content repeatedly. Your source will be less portable, and we do not support or recommend these techniques. But if you want a go anyway, see hints at www.sagehill.net/docbookxsl/DuplicateIDs.html. Note the reliance on xpointer(), and that the final technique is restricted to DocBook, a different XML vocabulary.

6.26 Accessibility

Continuing our discussion from Section 3.25 we begin by listing features of our conversion to HTML which happen automatically.

HTML Wherever possible we supply HTML elements and attributes that will be interpreted sensibly by a screen reader in the absence of the visual styling provided by CSS. This means we are cognizant of the role of headings (h1 through h6), provide HTML that passes validation checks, and so on. Employing attributes from the Accessible Rich Internet Applications suite of web standards (ARIA) at www.w3.org/WAI/ will go a long way to improving accessibility. This work is in-progress as of 2018-05-31.

Mathematics MathJax (mathjax.org) is the JavaScript library we use to render mathematics within the HTML output. It provides extensive capabilities for screen readers to render the mathematics audibly, and by default your project's output is configured to take advantage of these features. We refer the reader to the MathJax documentation of Accessibility Features at docs.mathjax.org for details. But here is a simple experiment you can do yourself right now to simulate how a blind reader could experience mathematics with the combination of PreTeXt, MathJax, and a screen reader.

1. Find some moderately complicated mathematics, such as in the "Mathematics" section of the sample article, or your own project, or the sample from MathJax copied below.
2. Bring up the context menu on that display (a mouse right-click for most).
3. Turn on the Accessibility > Explorer > Activate menu item. The page will reload, and the Explorer menu item will earn many more menu items. This setting is reasonably sticky, so you should not have to do this repeatedly. Having this on will incur some processing time as part of each page load, so you may want to turn it off later.
4. Turn on the Accessibility > Explorer > Subtitles menu item.
5. TAB until some mathematics is given focus (a discrete border appears).
6. SHIFT-SPACE will activate exploration of the mathematics with the Explorer. A subtitle, with an aural rendering of the mathematics, will appear below the display.

7. You can navigate (explore) the expression tree with the up, down, left, and right arrow keys. The subtitles will change as you do this.

From the MathJax demonstration page, Maxwell's equations for practice:

$$\begin{aligned} \nabla \times \vec{\mathbf{B}} - \frac{1}{c}\frac{\partial \vec{\mathbf{E}}}{\partial t} &= \frac{4\pi}{c}\vec{\mathbf{j}} \\ \nabla \cdot \vec{\mathbf{E}} &= 4\pi\rho \\ \nabla \times \vec{\mathbf{E}} + \frac{1}{c}\frac{\partial \vec{\mathbf{B}}}{\partial t} &= \vec{\mathbf{0}} \\ \nabla \cdot \vec{\mathbf{B}} &= 0 \end{aligned}$$

Skip to Main Content Repeatedly pressing the `Tab` key will move a reader from one location to the next in a web document. Since your Table of Contents in the left sidebar is a series of many links, a reader will need to tab through *all* of these to eventually reach the interesting content on a page.

However, we support a common device. The first link on every page is hidden from all readers, but an initial `Tab` will present a link labeled `Skip to Main Content` which when executed will take the reader past the Table of Contents and to the start of the content at the top of the page.

Colors We are sensitive to the fact that some readers have difficulty distinguishing between certain colors. So we do our best to distinguish text, or other elements, without relying exclusively on color. For example, the `<delete>` and `<insert>` elements may render text with strike-through and underlining (respectively) to show the distinction.

Here are features which are provided, but require your participation as the author.

Image Description Images you author or supply will be invisible to some readers. Within every `<image>` element you can use a `<description>` element. The content here will migrate to places like the HTML `@alt` attribute to be picked up by screen readers.

Make the content very expressive and detailed, but not overly verbose. Also, do not use any markup whatsoever, just simple characters, and avoid quotation marks. You can learn more at sites such as the one provided by the Web Accessibility Initiative (WAI) at Text alternatives for non-text content.

We cannot do this one for you, this is for the author only. But we can give you the tools do it as easily and as correctly as possible.

Image Formats University offices that provide services for students with disabilities are often interested in the images themselves from a text, as standalone files. For example, they might be able to manufacture tactile versions. You could use the `mbx` script to produce a variety of different formats and bundle these up in a single archive file for distribution at your book's website. Or you can make each image available through adjacent links placed automatically. We call these "image archives." See Subsection 6.11.4.

Cross-References Section 6.6 describes a variety of ways to customize the look and content of a cross-reference. You can create a larger target for clickable items by making the text as long as possible. So for example an `<xref>` authored as

```
<xref ref="theorem-FTC" text="type-global" />
```

would cause the clickable portion to be something like "Theorem 5.16", whereas

```
<xref ref="theorem-FTC" text="global" />
```

would then cause the clickable portion to be simply the much shorter "5.16". Of course, you can set a default style for your entire document, so it is not necessary to continually provide the `@text` attribute.

Watching a blind reader navigate a web page can be a very enlightening experience. Or you might even undertake learning one yourself. Here are some suggestions for getting started (current on 2018-05-31).

- NVDA, www.nvaccess.org, Windows, open source via GPL
- Orca, help.gnome.org/users, Linux, open source via LGPL
- VoiceOver, included with Apple's macOS and iOS
- ChromeVox, www.chromevox.com, ChromeOS, free from Google
- JAWS, www.freedomscientific.com, Windows, commercial

Chapter 7

(*) Add-Ons

Some features of the HTML version of a document rely on third-party services. Once an author configures them, then PreTeXt will do the rest for you. This chapter provides guidance on the configuration processes.

7.1 (*) Analytics

7.2 Search

Search facilities are enabled through Google Custom Search Engine. Please, please report any discrepancies in the following instructions as the setup interface at Google changes out from underneath us. These instructions are accurate as of 2016-12-12.

Besides being useful for search facilities, setting up a search engine might be a good way to alert Google of something newly available, and initiate your book's rise up the search results rankings.

1. Create an account with Google (GMail, YouTube, etc.) and make sure you are signed in.
2. Visit GCSE and `Add` a new search engine of follow `New Search Engine`.
3. Provide a URL for the top-level domain name/directory for your book/document. Everything below this will be indexed. We have taken some care to mark knowl content in a way compatible with the search facility, but there is more work to do here.
4. Give the engine a GCSE-specific name, so you can tell later which one it is when you have several.
5. Under `Edit Search Engine > Setup > Basics > Details > Search engine ID` find a string which uniquely identifies your new search engine. Save this, you'll need to make it part of your MBX document.
6. Under `Edit Search Engine > Setup > Admin` add co-authors or trusted backup personnel.
7. Under `Edit Search Engine > Business > Settings` set your `Advertising status` to the non-profit setting if you qualify (most universties should).
8. Fiddle with `Edit Search Engine > Look and Feel` at your own risk! Only the defaults are tested and supported.

9. `Edit Search Engine > Setup > Indexing` sends you to Google Search Console to see if your book is already being indexed. YOu may need to go through a confirmation process to establish that you are the owner of the website being indexed. If you see taht your book is not yet being indexed, you may want to wait as long as a week before your material does get indexed and you make a search box available.

List 7.2.1: Configuring Google Custom Search

1. The `Search engine ID` you saved from above is referenced in Google's code as a `cx` number. Add an element in your MBX source as `docinfo/search/google/cx` with the value of your book's ID as the content. See the sample article for a working example to mimic.
2. The `cx` element will alert the PreTeXt conversion and fully enable and implement search. You're done, and everything should just work. You should see a Google-branded search box to the top right of each of your pages. (We have no control over the branding.)
3. Time to rebuild your official HTML output and make the improved version available.

List 7.2.2: Configuring PreTeXt for Google Search

7.3 (*) Annotation

Chapter 8

Authoring Advice

In this chapter we gather advice on authoring. In some cases it is discipline-specific.

8.1 Writing Your Student-Friendly Math Textbook

Kathy Yoshiwara

So you are writing a math textbook. You love your subject enough to put in the hours, and you probably have some ideas on how the standard presentation can be improved. You care about good pedagogy and want to engage your students.

You know that writing a text for undergraduates requires a different style from writing a research paper or a scholarly article for colleagues. But how to achieve that style? A good first step is to adjust your linguistic goal from

Elegant Argument to Illuminating Explanation

We can examine this strategy in three areas:

1. Language
2. Layout or Format
3. Content

Language. In the exposition, it is useful to adopt an informal voice. But, perhaps counter-intuitively, that does not mean a conversational voice. In conversation, you have gestures and tone of voice to help convey meaning: what are the main points, what are helpful hints, what are asides, and what are social interactions.

None of these prompts are available when communicating in print. The author must shape the material so that the reader can navigate the ideas on his or her own. It is best to be as brief and direct as possible. After writing a section or so, go back and omit any unnecessary words or phrases. For example, before presenting an Example, there is no need to say "Here is an example to illustrate the ideas we have just discussed." Subconsciously, the reader must absorb and then jettison this comment as unimportant. Doing so constantly leads to "reader fatigue."

Often an explanation can be improved just by (judiciously) making it shorter. In addition, while trying to understand a new concept, most students will not find helpful our philosophical musings or historical anecdotes. These can be presented in a sidebar or addendum. Do try to relate new ideas to previous ones and show how they fit into the overall scheme, but avoid references such as "we'll need this for our later study of (whatever)."

The principles of good writing for any format apply equally to textbooks, mathematical or otherwise. Usually, active voice is more effective than passive voice, and a positive form is clearer than a negative one. Strunk and White's classic *The Elements of Style* and, more recently, George Gopen's reader expectation

approach are standard resources for techniques of composition. (Although sometimes their advice is contradictory, which just goes to show that no rule is appropriate in every situation.) Steven Pinker's *The Sense of Style* is also useful.In a textbook, it is good practice to deliver material in digestible portions. Try to keep blocks of uninterrupted text rather short. Break up the exposition visually with boxes, Examples, Cautions, Notes, and so on. Use bulleted or numbered lists to highlight important points. Consider whether it would be more effective to start a particular section with a motivating example, or perhaps with a few sentences explaining how the new topic arises naturally, or in some other way.

Instead of opening your textbook with a "Chapter 0" type catalog of all the notation and terminology needed in the entire book, it would be kinder to students to introduce new notation and terminology as needed. Also, it is not necessary to front-load all the information about a topic at one time; let students absorb and practice the fundamental ideas, then return later to elaborate, generalize, or present exceptions. Resist the temptation to proceed too quickly to general or abstract statements. Most people grasp abstract ideas more easily if they first see specific and concrete examples.

Choosing effective examples helps make your text student-friendly. A simple example has greater impact than a complicated one. If you are introducing asymptotes, the graph of

$$y = \frac{x-3}{x-2}$$

is a better example than

$$y = \frac{x^2-9}{x^2-x-2}.$$

It is also better than

$$y = \frac{1}{x}$$

because the latter may lead students to believe that asymptotes are identified with the coordinate axes. If you are introducing subgroups, do not limit your examples to subgroups of cyclic groups. Always consider your example from the students' point of view! Is your example too general or too specific? Are there confounding features that may distract from the intended message?

What about proofs? These days a good background in mathematics is necessary for a wide variety of occupations, so that only a small number of your students may be headed to graduate school in mathematics, even in junior or senior level classes. Here is a good place to strive for "illuminating" rather than "elegant."

Content. Choosing and organizing the material for your textbook requires more thought than any other aspect of the creative process. To make your textbook truly student-friendly, you may want to rethink the traditional or standard order of content. If you are used to presenting material by topic, you might want to consider how better to make connections or to take advantage of sound pedagogical principles.

For example, forty years ago high school algebra was taught by first covering all the "one-variable" material and then (if time permitted) considering graphs and other "two-variable" topics. Now it is considered more effective to study graphs throughout the course. Linear algebra courses used to start with the study of vector spaces. Now many texts prefer to begin with systems of linear equations. In trigonometry, maybe we can start with just three trig functions instead of six; maybe we can work in one quadrant first, then add the second quadrant, before treating all four.

In a sense, textbooks drive the curriculum. Think of the innovations introduced in the calculus renewal movement that are now incorporated into most calculus texts: the catalog of functions, the rule of four, the inclusion of conceptual exercises. Many instructors, especially adjunct or part-time instructors, rely on their textbooks to shape their courses. You have a real opportunity to influence the direction of your field and to shape the way it is taught. Now get to work.

Chapter 9

The `mbx` Script

XSL is a very powerful language for text processing. However, it cannot do everything. The `mbx` script is a Swiss Army Knife of sorts to operate on parts of your document and manage processing that requires the application of external programs, such as LaTeX and Sage.

9.1 Running `mbx`

`mbx` is a Python script, so you will need to have the Python interpreter on your system. At a command-line (in a terminal or console), you can run

```
mbx -h
```

to get a summary of the commands. Some of the processing may take a long time, or you may experience trouble. There are two switches to enable more verbose output in your terminal or console.

```
mbx -v
```

will provide progress indicators, while

```
mbx -vv
```

will provide progress indicators along with additional information that will help you or a fellow author to discern where a problem lies. Please *include all of this output* if you are asking for help, and do not assume you know exactly which part is relevant.

9.2 Example Use

Here is a typical example of using `mbx`. You have several (or many!) diagrams and figures in your PreTeXt source, all authored in the TikZ language, and so packaged up within `<latex-image-code>` elements. Your LaTeX/PDF output looks beautiful, since PreTeXt simply inserts the TikZ code into the right place in the generated `*.tex` file, and you have done this several times until your figures look just right.

Now you need to generate the SVG versions of your images that will accompany your HTML version of your book and provide nice scalable graphics. This is exactly the sort of chore the `mbx` script was designed for. You might run

```
mbx -vv -c latex-image -f svg -d ~/books/aota/images ~/books/aota/animals.xml
```

Here `-c` is specifying the "component" of your book to process, and `-f` is specifying the "format" of what is being produced. The `-d` argument specifies a directory where the output ends up, in this case a collection of SVG files, one per image.

9.3 Strategy

Much like the build advice at the end of Section 4.8, the `mbx` script collects necessary bits into a system-created temporary directory, does its work, and copies out the desired results. So in the example of the previous section, each chunk of TikZ code is isolated, your LaTeX macros are copied from `<docinfo>`, and a syntactically correct LaTeX file is produced (one per image). Then `mbx` calls your LaTeX executable on each of these files to produce a one-page PDF. This is then cropped and converted to an SVG version, which at the end is copied to the location specified in the `-d` argument.

Some insight into failures can be found in the temporary directory where all this processing happened. (We leave the directory, and its contents, behind for the system to clean-up later). Early in the `-vv` doubly-verbose output, this directory is reported after the string `temporary directory:`.

Some notes:

- If you have modularized your source across more than one XML file, then be sure to provide your "top-level" or "master" file as the final argument to the script, just like you would for an invocation of `xsltproc`. It is important to understand that your source is one huge "source tree" and your file-by-file modularization is never respected or recognized in any way. In particular, use of the `xinclude` mechanism is handled by the script, and you should not apply the script to each of your source files individually. If image production (or some other task) takes a long time, see Section 9.5 for a way to have the script restrict its action to only a portion of your project.
- Certain arguments that are filenames require a full (not relative) path to locate the right objects. In the example above, you can see the source file requires this (where the shell in use here will expand the ~ to the user's home directory). The `-d` flag does not require a full path, and so can be specified relative to the current working directory.
- Do not place the script, or configuration files, anywhere else (except as recommended for your personal copy of the configuration file). The locations are important for locating other files, such as the stylesheets used to isolate parts of your project for processing.
- Much of the work of this script happens in the temporary directory described above. We leave a lot of intermediate work behind in this directory. Often, exploring this directory is helpful when debugging problems, or a failure to finish successfully.

9.4 Debugging Image Generation

A principal use of the `mbx` script is to isolate source code from `latex-image-code` sections, package them up as proper `*.tex` files, run LaTeX to make cropped PDF versions, and then convert these to other formats such as SVG or PNG.

Much of this activity happens in a temporary directory. If you use the `-vv` switch described above, then one of the messages early in the output describing the initialization will be the name of this directory. Looking to see what files end up there, and what those files contain, is often useful in determining the step where this toolchain fails, and maybe even why.

Another option is to ask for the actual `*.tex` files as the result of a run. This is accomplished with the `-f source` option when invoking `mbx`. If the right packages or macros are not being employed in these files, this is an easy way to get at the source files for inspection and analysis.

9.5 Restricting the Scope

The `-r` (`--restrict`) switch deserves special mention. It is followed by the value of an `@xml:id` attribute present in your source XML file. Then whatever action the script is asked to perform, it will only act on a subtree of the hierarchy, rooted at the element with the given `@xml:id` value.

So if your images are complex or numerous (or both!) and take a long time to process, you can restrict attention to whatever part of the document you are actively editing, and you can even restrict to a single `<image>` and so produce just a single graphics file.

9.6 Configuring External Helper Programs

Our main processor, `xsltproc`, is not a general-purpose compiler, and does not "call" external programs. That is the primary purpose of the `mbx` script. You will see a configuration file, `script/mbx.cfg`, as part of the distribution. Read the comments at the top of this file, but foremost, realize that you are not meant to edit this file. It is a template, and any changes you make will be overwritten with an original version when you update. Instead, make a copy and place it as `mathbook/user/mbx.cfg`. The script will look for this copy first, then fallback to the generic version.

The entries of this file are the names of executable files that perform certain tasks as part of the script's functions. If it seems that certain helper programs are not being found, you can provide full path names, and that may solve the problem.

9.7 Output

Once the `mbx` script produces results, it is your responsibility to keep track of them. Early in a project, you may wish to regenerate images regularly, and not save the results permanently (for instance, you may not want to track them under version control). In this case, a build script is very useful. For a mature project, you may only regenerate images you know have changed, or when you create a new edition. And you may wish to include them with your source files, under revision control, for your readers to use.

More critically, you need to place items generated by the `mbx` script where the rest of your output can find them. For example, by default HTML output expects images to be in a subdirectory named `images`. (This default can be changed, though it seems almost nobody does.)

9.8 `mbx` Capabilities

Again, the command `mbx -h` will remind you of the various options for the script. The followinng is a brief summary, in general terms, of what is possible.

LaTeX Graphics LaTeX has a variety of languages for specifying images, such as xypic, pgfplots, and TikZ. By including the necessary pacakges or setup commands in `docinfo/latex-preamble`, these can all be generated at once, in the manner of the example earlier.

Asymptote Images described by the Asymptote language can be processed in a manner entirely similar to that for images described with LaTeX graphics languages. This requires having the `asy` executable on your system and locatable via the `mbx` configuration file.

Sage Plot If you have a version of Sage installed on your system, you can specify the path to the executable and obtain images described by Sage code. See the sample article for more information.

All Formats If you desire images in a wide variety of formats, the option `-f all` will oblige.

YouTube Thumbnails For each YouTube video (or itemized playlist) you specify, the script will go the YouTube site and grab a thumbnail image for that video (or first video from the itemized playlist). These get used in static formats, such as PDF.

WeBWorK Various conversions of WeBWorK problems are facilitated through communication with a WeBWorK server. This server is specified as an argument to the `-s` option. See Chapter 10 for the details of this procedure.

9.9 `mbx` on Windows

At present, the `mbx` script assumes that your installation is similar to the one described in Appendix H. Making the `mbx` script Windows-compatible is an ongoing project. It is possible you may find a bug, which we would ask that you report. In addition, if you have followed the directions in Appendix H, then you will

need to customize the mbx.cfg configuration file, which tells mbx where to find the helper programs it relies on. See below for the details.

The script uses a LaTeX utility called pdfcrop to, well, crop PDF images generated by a LaTeX engine. This utility has not been made to work in the Windows CMD shell. If you want to generate images on Windows, you should use the Git Bash shell as described in Section H.3.

1. Copy the file

   /path/to/mathbook/script/mbx.cfg

   to

   /path/to/mathbook/user/mbx.cfg

2. Replace the right-hand side of the entry for pdfpng with the full path name of the convert utility installed with ImageMagick (Section H.5), ***using forward slashes***. It will be something like

   c:/ImageMagick-7.0.1-Q16/convert.exe

9.10 Python requests Library

In some situations the mbx script will go out on the Internet to fetch some interesting bits for you, saving you the trouble. These include

- Grabbing, downloading, and organizing stock thumbnails for YouTube videos, using a standard API provided for this purpose. These get used in PDF output in place of embedded videos.
- WeBWorK problems that are stored on a remote server will give up LaTeX versions if asked. Again, these are used for PDF output in place of interactive versions.

We use the Python requests module/library to manage the connections to these external servers. There are two items to be aware of.

Installation. This library is not available on Apple computers by default. From Alex Jordan comes the following incantation at the command line,

```
sudo easy_install pip
sudo pip install requests
```

which was last tested 2017-03-31. Please update us if the situation has changed or there is more to add here.

Warnings. Using this library to connect to a webserver securely via HTTPS will raise a warning since the support for SSL certificates is not complete. You will see messages similar to

```
site-packages/requests/packages/urllib3/connectionpool.py:843:
InsecureRequestWarning: Unverified HTTPS request is being made.
Adding certificate verification is strongly advised.
See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
InsecureRequestWarning
```

We have not figured out the best way to stop these, as of 2017-04-01.

Chapter 10

WeBWorK Automated Homework Problems

Alex Jordan

With a WeBWorK server (version 2.14 or higher, or webwork-ptx.aimath.org) and a little setup work, you can embed WeBWorK exercises in your PreTeXt project. HTML output will have interactive problem cells. PDF output will contain static versions of exercises. And all such exercises can be archived by the `mbx` script into a file tree to be uploaded onto the WeBWorK server for use in the "traditional" way.

10.1 Configuring a WeBWorK Course for PreTeXt

We assume a mild familiarity with administrating a WeBWorK server. The version of WeBWorK needs to be 2.14 or later for use with PreTeXt. Using the `admin` course, create a course named `anonymous`. In the course's Course Configuration menu, set all permissions to `admin` (or perhaps set some to the even more restrictive `nobody`). Except set "Allowed to login to the course" to `login_proctor`.

In the Classlist Editor, add a user named `anonymous`, and set that user's permission level to `login_proctor`, the permission level one higher than `student`. Set that user's password to `anonymous`. Note that because this is public information, anyone will be able to log into this course as user `anonymous`. This is why setting the permissions earlier is very important. (Especially preventing this user from changing its own password.)

Add the following lines to the `course.conf` file (which lives in the parent folder of the `templates/` folder.)

```
# Hide message about previewing hints and solutions for instructors
$pg{specialPGEnvironmentVars}{ALWAYS_SHOW_HINT_PERMISSION_LEVEL} = 100;
$pg{specialPGEnvironmentVars}{ALWAYS_SHOW_SOLUTION_PERMISSION_LEVEL} = 100;
```

In the `templates/macros/` folder, edit `PGcourse.pl` (or create it if need be) and add the lines:

```
#### Replace essay boxes with a message
sub essay_box {
    my $out = MODES(
        TeX => '',
        Latex2HTML => '',
        HTML => qq!<P>If you were logged into a WeBWorK course
        and this problem were assigned to you,
        you would be able to submit an essay answer
        that would be graded later by a human being.</P>!,
        PTX => '',
    );
    $out;
```

```
};

#### Suppress essay help link
sub essay_help {};

#### How many attempts until hint is available
$showHint = -1;
# May be a bug that WeBWorK requires -1 instead of 0
# for immediate access to hints

1;
```

Now PreTeXt will be able to communicate with this course to retrieve what is needed.

10.2 WeBWorK Problems in Source

A `<webwork>` tag must be inside an `<exercise>`, optionally preceded by an `<introduction>`, and optionally followed by a `<conclusion>`.

```
<exercise>
    <introduction>
    </introduction>

    <webwork>
    </webwork>

    <conclusion>
    </conclusion>
</exercise>
```

There are several methods for putting content into the `<webwork>`. (Note that an empty `<webwork>` with no attributes will simply produce the camelcase WeBWorK logo.)

10.2.1 Using an Existing WeBWorK Problem

If a problem already exists and is accessible from the `anonymous` course's `templates/` folder, then you can simply include it as a `@source` attribute. For example, if it is a problem in the Open Problem Library (OPL) then relative to the `templates/` folder, its path is `Library/...` and you may use:

```
<webwork source="Library/PCC/BasicAlgebra/Exponents/exponentsMultiplication0.pg" />
```

Or if you have a problem's PG file, you can upload it into the `anonymous` course's `templates/local/` folder and use it with:

```
<webwork source="local/my_problem.pg" />
```

10.2.2 Perl-free Problems

If you'd just like to rattle off a quick question with no randomization, you can do as in this example:

```
<exercise>
    <webwork>
        <statement>
            <p><m>1+2=</m><var name="'3'" width="5" /></p>
        </statement>
    </webwork>
</exercise>
```

The <exercise> above could be given an optional <title>, <introduction>, and <conclusion>. The <webwork> inside could be given a <hint> and <solution>. These are discussed in Subsection 10.2.3.

In the above example, '3' is the @name attribute to a <var> element. This is how to create an answer blank that is expecting 3 as the answer. What you give as a @name attribute will be passed to PG's Compute() command, so it needs to be valid input for Compute(). Note that you could pass a string encased in quotes, or a perl expression. Just be mindful of the difference:

- 8**2 will process a perl real using exponentiation and lead to the MathObject Real 64.
- '8^2' will process a perl string and lead to the MathObject Real 64.
- 8^2 will process the perl real using bitwise XOR and lead to the MathObject Real 10.

The default context is Numeric, which understands numerical expressions and formulaic expressions in the variable x. You can activate some other context as in this example:

```
<exercise>
    <webwork>
        <setup>
            <pg-code>
                Context("ImplicitPlane");
            </pg-code>
        </setup>
        <statement>
            <p>The answer is <m>x+y=1</m>.</p>
            <p><var name="'x+y=1'" width="8" /></p>
        </statement>
    </webwork>
</exercise>
```

Many special contexts are automatically detected by PreTeXt, and it loads the appropriate macro file into the PG problem. However you may need to explicitly load a macro file as described in Subsection 10.2.3.

10.2.3 PG code in Problems

To have randomization in problems or otherwise take advantage of the algorithmic programming capabilities of Perl and WeBWorK's PG language requires using a <setup> tag. Having at least a little familiarity with coding problems in WeBWorK is necessary, although for simpler problems you could get away with mimicking the sample article in mathbook/examples/webwork/. A <statement>, (optional) <hint>, and (optional) <solution> follow.

```
<webwork>

    <setup>
    </setup>

    <statement>
    </statement>

    <hint>
    </hint>

    <solution>
    </solution>

</webwork>
```

The <setup> contains a <pg-code>. If you are familiar with code for WeBWorK PG problems, the <pg-code> contains lines of PG code that would appear in the "setup" portion of the problem. Typically, this is the code that follows TEXT(beginproblem()); and precedes the first BEGIN_TEXT or BEGIN_PGML. If your code needs any special WeBWorK macro libraries, you may load them in a <pg-macros> tag prior to <setup>, with each such .pl file's name inside a <macro-file> tag. However many of the most common macro libraries will be loaded automatically based on the content and attributes you use in the rest of your problem.

Here is a small example. Following the example, we'll continue discussing <statement> and <solution>.

```
<webwork>
    <title>Integer Addition</title>

    <setup>
        <pg-code>
            $a = Compute(random(1, 9, 1));
            $b = Compute(random(1, 9, 1));
            $c = $a + $b;
        </pg-code>
    </setup>

    <statement>
        <p>Compute <m><var name="$a" />+<var name="$b" /></m>.</p>
        <instruction>Type your answer without using the <c>+</c> sign.</instruction>
        <p>The sum is <var name="$c" width="2" />.</p>
    </statement>

    <solution>
        <p><m><var name="$a" />+<var name="$b" />=<var name="$c" /></m>.</p>
    </solution>
</webwork>
```

Within a <statement>, <hint>, or <solution>, reference <var> tags by @name.

Within the <statement>, a <var> tag with either a @width or @form attribute creates an input field. The @name attribute declares what the answer will be.

A <var> can have @form="essay", in which case it need not have a @name attribute. This is for open-ended questions that must be graded by a human. The form field will be an expandable input block if the question is served to an authenticated user within WeBWorK. But for the WeBWorK cells in PTX HTML output, there will just be a message explaining that there is no place to enter an answer.

A <var> can have @form="array". You would use this when the answer is a Matrix or Vector MathObject (a WeBWorK classification) to cause the input form to be an array of smaller fields instead of one big field.

A <var> can have @form="popup" or @form="buttons" for multiple choice questions.

If you are familiar with PG, then in your <pg-code> you might write a custom evaluator (a combination of a custom answer checker, post filters, pre filters, etc.). If you store this similar to

```
$my_evaluator = $answer -> cmp(...);
```

then the <var> can have @evaluator="\textdollar{}my\textunderscore{}evaluator".

An <instruction> is specific instructions for how the reader might type or otherwise electronically submit their answer. Contents of an <instruction> will be omitted from print and other static output forms. The <instruction> is a peer to <p>, but may only contain "short text" children.

Some general information on authoring WeBWorK problems can be found in a set of videos at

webwork.maa.org/wiki/Problem_Authoring_Videos

Not all of this is relevant to authoring within PreTeXt but there are parts that will be helpful for constructing the Perl code necessary for randomized problems.

10.2.4 Reusing a <webwork> by @xml:id

Planned.

10.3 Processing

10.3.1 Extraction and Merging

A PreTeXt project that uses WeBWorK must first have its WeBWorK content extracted into an auxiliary XML file before anything else can be done. Use the `mbx` script to extract PreTeXt content from the WeBWorK server into a *folder*, which you might call `webwork-extraction/` as in this example:

```
$ mbx -c webwork -s <server> -d webwork-extraction <xml>
```

The WeBWorK server needs to be version 2.14 or later, specified with its protocol and domain, like `https://webwork-ptx.aimath.org`. (If you do not have a server, you may use `https://webwork-ptx.aimath.org`.)

You may need to specify a path to the `webwork-extraction` folder. Any image files that the WeBWorK server generates will be stored inside this folder. An auxiliary XML *file* called `webwork-extraction.xml` will be created in this folder. (Note that you can name the folder whatever you like, but the auxiliary file that is created will always be named `webwork-extraction.xml`.)

Next, use `xsltproc` with `pretext-merge.xsl` to merge your entire source tree with the extracted WeBWorK content. The string parameter `webwork.extraction` must identify the auxiliary XML file created in the previous step. Store the output in some file, for example `merge.ptx` in this example:

```
$ xsltproc --stringparam webwork.extraction webwork-extraction.xml pretext-merge.xsl <xml> > merge.ptx
```

Note that you may need to provide file paths to `webwork-extraction.xml` and `pretext-merge.xsl`.

10.3.2 HTML output

When you execute `xsltproc` using `mathbook-html.xsl`, apply it to the merged file described above, not your original source. For example:

```
$ xsltproc mathbook-html.xsl merge.ptx
```

There are several string parameters you may pass to `xsltproc`.

stringparam	options
`webwork.inline.static`	
	`'no'` (default) means inline exercises render as interactive.
	`'yes'` means inline exercises render as static.
	`'preview'` (planned) means inline exercises render as static until you click to activate them.
`webwork.divisional.static`	
	`'no'` means divisional exercises render as interactive.
	`'yes'` (default) means divisional exercises render as static.
	`'preview'` (planned) means divisional exercises render as static until you click to activate them.
`html.knowl.exercise.inline`	
	`'no'` means inline exercises appear on page load.
	`'yes'` (default) means inline exercises are hidden in knowls.
`html.knowl.exercise.sectional`	
	`'no'` (default) means divisional exercises appear on page load.
	`'yes'` means divisional exercises are hidden in knowls.

10.3.3 LaTeX output

When you execute `xsltproc` using `mathbook-latex.xsl`, apply it to the merged file described above, not your original source. For example:

```
$ xsltproc mathbook-latex.xsl merge.ptx
```

One string parameter you can pass to `xsltproc` is `latex.fillin.style`, which can take values `'underline'` (the default) or `'box'`.

10.3.4 Creating Files for Uploading to WeBWorK

All of the `<webwork>` that you have written into your project can be "harvested" and put into their own `.pg` files by the `mbx` script. These files are created with a folder structure that follows the chunking scheme you specify. This process also creates set definition files (`.def`) for each chunk (say, for each section): one for inline exercises and one for divisional exercises. For `<webwork>` problems that come from the WeBWorK server, the `.def` file will include them as well. This archiving process creates set header `.pg` files for each set definition.

As with other WeBWorK processing, you must use the extraction and merge process first that is described in Subsection 10.3.1. For example:

```
$ xsltproc --stringparam chunk.level 1 pretext-ww-problem-sets.xsl merge.ptx
```

This creates a folder named after your book title, which has a folder tree with all of the `.pg` and `.def` files laid out according to your chunk level. You can compress this folder and upload it into an active WeBWorK course where you may then assign the sets to your students (and modify, as you like).

Appendix A

Welcome to the PreTeXt Community

Thank-you for your interest in PreTeXt, and welcome! This appendix is meant to answer some questions you may have about how this open-source project is organized, how you can get help, and how you can contribute back.

PreTeXt is not LaTeX and it is not Word. Some "features" of those languages are intentionally missing, but more importantly, the mind-set expected of authors is completely different. One of the most important distinctions is that PreTeXt treats the author, the publisher, and the reader (which might be an instructor) as separate entities—even when those are all the same person.

Newcomers to PreTeXt should focus on the author role. Make sure the structure of your book is marked-up properly. If something looks wrong in your output, assume it is a problem with the source markup. If the source is correct but the output does not look as you wish, leave that as a problem for the publisher, to be addressed *after* you have finished writing the content.

A.1 Help and Support

There is a documentation area at the project website. Presumably, that is where you found this *Author's Guide*. This is what another software project might call the *User's Manual*. So start here. Re-read Section 1.1 on the project philosophy and the principles in List 1.1.1 at regular intervals. Chapter 3 is meant to inform you on the features of PreTeXt, without getting into all the details. It will frequently refer you to Chapter 6 for all those details.

PreTeXt is fundamentally a specification of a set of elements and attributes, a topic discussed in Chapter 5. This chapter also discusses validation. Once comfortable, but before authoring lots of material, take the time to get validation working, and use it regularly. Do not save it for last.

There are many examples available in an area on the website. Compare PreTeXt source to the resulting output (in both directions). The "sample article" is not always pretty, since it is used for testing, but it does try to have one of everything.

When the above is not sufficient, the `pretext-support` Google Group is the right place to ask questions. If you are trying to determine which elements to use to accomplish something, provide some context. Do not ask, "How do I print a line of text upside-down?" Instead say, "I am writing a monograph on mammalian vision, and I'd like the reader to use a mirror to view a line of text written upside down. What is the best way to do that?" (I think the answer would be: make an image and include that, rather than trying to get reflected text.) Sometimes a quick search of the Goggle Group may yield insights.

When you have a question about design, or are pretty sure you have encountered a bug, then the `pretext-dev` list is the place to start a discussion. As you get more comfortable with PreTeXt this may be a more useful venue for more detailed discussions.

A.2 Feature Requests and Reporting Problems

We use the **issue tracker** at the GitHub site as an organized to-do list. You do not need to know anything about `git` to use this forum. Just make a new issue, or make a comment on an existing issue. Frequently, a discussion on the groups will culminate with the creation of a new issue. It is nice to have a link from the discussion to the issue, and vice-versa.

If you are asked to create an issue in response to a discussion you initiated, please consider doing so. It will save the other volunteers just a bit of time to work on other parts of PreTeXt. And you will also get email-for-life as the issue is discussed and eventually closed. It is a very helpful contribution.

As you gain more experience, you will identify bugs, obsolete instructions, typos, etc. Search the issues to see if there is something relevant you can add to. For example, your particular version of a bug might provide the key insight into identifying the cause. When you are certain something is wrong, and there is no need to discuss it in the groups, feel free to go straight to making an issue. For something like a list of typos, make a single issue and just keep editing your initial post.

If you have some structural problem, see if you can reproduce it by adding into the minimal example, and post that *entire* source file. If the rendering in HTML is a bit off, be sure to post a link to a *live* example, not just a verbal description and definitely not a screenshot.

A.3 Contributing

As a project that is licensed openly, we welcome contributions. And this does not necessarily mean you need to learn our primary language, XSL. For ideas, find the issues on GitHub that have the label `contributor project`. As one example, it would be very helpful if a member of the community would create and maintain a Wikipedia page. That is a skill that is very distinct from the other skills used to create and maintain other parts of the project. See Issue #207.

The documentation is authored in PreTeXt, so you know how to create additions, clean-up obsolete parts, and fix typos. If you know `git` and GitHub, then a pull request (on a new branch!) is a very economical way for us to manage contributions. Even if you do not know GitHub, we can easily accept files written in PreTeXt that contain changes to the documentation. (Send them to us by email, or post as attachments.)

Conversions are written in XSL, a declarative language. It has a steep learning curve, but is very powerful for an application like this. Start small, and we do not mind helping you along with suggestions and critiques. Do not begin an ambitious task unless your skills are up to it.

A.4 Personal Email

How do we put this politely? Personal email to the core PreTeXt developers should be your last resort. We are not unfriendly—just the opposite. We would love to hear from you, but in the groups. Here is the rationale:

- You may get a better answer from somebody who is not the most active developer, but understands your particular need better than anybody else. You will never get that answer with a personal email.
- Developers teach university courses, travel to professional meetings, sleep at night, take naps, turn off their email for big coding pushes, and sometimes travel in the wilderness and are offline for days at a time. You are likely to get quick responses from the core developers through the groups, but if they are not available a personal email may get a slower response. (I am inclined to answer posts on the groups before I work through personal email.)
- Many contributors prefer to provide help in a public forum because then their efforts are more widely recognized.
- Your question, and its answer, are searchable by others. (The groups and issues are public.) A personal email is no help to anybody else.

- We prefer to make as many decisions as possible *openly*. So a discussion on a public group or site is there for all to see, now and later.
- We depend on granting agencies for much of our funding. Membership in groups, forks on GitHub, activity in the groups, number of issues, and number of contributors to the repository, are all crude measures of the health of the project. Personal emails add nothing to those measures.
- Everybody who has committed their big writing project to PreTeXt likes to see an active, responsive, friendly community supporting its use and growth. Just by asking a necessary question, you can add to that community.

We understand that nobody likes to pop their head up and ask a "stupid question." But it is counter-productive to do personally what we can do better collectively. The groups are friendly forums (we will enforce that if we ever have to) and everybody there made an initial post once. And the group members largely enjoy sharing their advice, experience, and knowledge. So, please make a contribution simply by saving the personal emails for that which really is personal. Thanks, and we'll look forward to chatting with you on the groups!

Appendix B

FAQ: Frequently Asked Questions

Technical questions first, followed by questions about markup.

On my local machine, why are the knowls/Sage cells empty? Viewing a file on your computer is not the same as visiting a web page. There is no web server, so you should assume that things will be different.

The cause of the difficulty is that some browsers (Chrome and Safari, for example) will not fetch the content of knowls when viewing a file on your local computer. Those browsers consider opening a local knowl to be a security risk. As of the time of this writing, Firefox will open those knowls.

Your options are to use Firefox, to transfer the files to an online server, or to set up a local web server so that local files operate in the usual way.

To set up a local server:

1. In a terminal, go to the directory containing the HTML files.
2. Do: `python -m SimpleHTTPServer`
3. Note the port indicated in the output (let's assume it is `8000`)
4. Go to `http://127.0.0.1:8000/` to see a listing of the files. Click on a file and you will be viewing it exactly as if it was on the Web.

The instructions above are for Python 2. For Python 3, as well as further discussion, see the Author's Guide.

How do I install `xsltproc` on Ubuntu or Debian Linux? `sudo apt-get install xsltproc`

How do I install `pdf2svg`? `pdf2svg` is necessary for TikZ diagrams in HTML.

On Debian or Ubuntu Linux: `sudo apt-get install pdf2svg`.

To install `pdf2svg` on a Mac, you will need to install MacPorts. Read the directions carefully, since you will need to install Xcode (available from the Mac App Store) first. Make sure that the command line tools are installed by running Xcode. After Xcode is installed read the directions to install Macports. Once MacPorts is installed run the following command to install `pdf2svg`: `sudo port install pdf2svg`. Be patient as this will take a few minutes. To get rid of any intermediate build files, run the command `sudo port clean --all all`. Again be patient.

If you have trouble with MacPorts, try HomeBrew.

Why is there no tag for bold? Because the first principle of PreTeXt is that the markup captures the structure of the document, not the appearance.

A better question is: *why* do you want to print something in bold? Is it emphasis? (See `<em>`.) Is it the volume number of a journal? (See `<journal>`.) Do you want to SHOUT? Try `<alert>`. And so

on. There are lots of good answers, some of which are not yet implemented. We would love to hear about elements you need that are about expressing content, and not about altering presentation. See Principle 1.

Why do I get no output and some warnings about bugs? There is a good chance you have "modularized" your source files and have not included the `--xinclude` switch on the command-line when you ran `xsltproc`. (See Section 4.2.) If that was your mistake, then you should have seen a warning. Please check to see if there was a warning you missed. (If not, we can improve the warning if you tell us how your source was organized. So please do, since we would love to hear about it.)

I don't like surprises and have not updated in months. Why do I now have a problem? We almost never release mistaken code that breaks output produced from valid source (Chapter 5). And when we have, the cause is usually a small typo, or something that gets fixed easily and quickly. We do sometimes make backwards-incompatible changes, but you always get warnings, often the changes can wait, there are announcements on the mailing lists, and whenever possible, we update tools that automate the changes.

As volunteers, we cannot support problems from old versions, and sometimes we have to implement changes in reaction to third-party software (like MathJax), which is beyond our control. So while development is rapid, we implore you to remain on the `dev` branch of the repository and `git pull` regularly. Such as *daily*, with your morning coffee as you sit down to write, or with your last compilation of the evening. You will have *far fewer unpleasant surprises* this way, and we can help you better.

We understand that PreTeXt is a moving target, but we are iterating to a better state, and it is best to have everybody along for the *whole* ride.

How do I put mathematics into my list labels? First, realize that the way LaTeX uses the term **label** in the context of lists is different from how much of the rest of the world uses the term in this context. In our case the `@label` attribute describes the style of the grouping markers. For example, bullets versus squares on items of an unordered list, or Roman numerals versus Arabic numerals on an ordered list. So this attribute conveys information *about* the list, not content *of* the list. And even if you tried putting an `<m>` tag into the value of the attribute, you would not have any luck since XML does not even allow that construction. Finally, there is no real good way to accomplish this in HTML, so conversion to that format would be difficult.

The alternative is to use a description list, with tag `<dl>`. You are reading one right now. Put your mathematics in the `<title>` and the associated content into the remainder of the `<li>`.

I have errors when I try to validate my markup, but everything looks okay when I make the HTML and th
Occasionally the schema lags behind the code, so your first step is to post to `pretext-support@googlegroups.com` to find out if it is a problem with the schema.

Possibly there is another way to accomplish what you want, and that markup will fit the schema. Or maybe what you are doing meets the "common and reasonable" test and can become a feature request. The discussion on `pretext-support` should provide an answer.

Why are theorems, definitions, examples, remarks, etc. all numbered using the same counter? The following is an argument in favor of using common counters for blocks of similar appearance. The argument is stronger in the context of using a printed copy of the book, where physical page flipping is necessary, but also applies to scrolling through a (long) page in a web browser.

Suppose your math professor gave you a note to review Theorem 2.4.7 in your textbook. The "2.4" is useful information directing you to Chapter 2, Section 4. You can tell by the "7" that the theorem is probably not right at the beginning of the section, so you open to the middle of the section. You find yourself on a page with no theorems, but you do see Example 2.4.11. What do you do: flip forward or flip backward?

If theorems and examples are numbered using separate counters, you have no information about which way to go. You need to make a random decision, and flip pages until you find another theorem that you can use as a guidepost. And theorems may be rare and sparse, so it may take quite a bit of page

flipping to find that guidepost. You may end up at Theorem 2.4.8, telling you that you need to flip backward now. But how far? Will it be one page earlier or twenty?

If theorems and examples are using the same counter, then you know that you need to flip backward. And perhaps more importantly, the oblivious reader who thinks they are looking for "2.4.7" (and is not thinking about "Theorem") sees the "2.4.11" and correctly flips backward without even realizing the potential for different counters. As they pass Definition 2.4.10 and Example 2.4.9, they have a sense of the pace at which they are converging on Theorem 2.4.7. This is the main argument for the use of common counters: it makes everything easier to locate.

For another point, perhaps you have read a math book in the past and have seen something like "and according to 2.4.7...". What if the book has Theorem 2.4.7 and also Example 2.4.7? Which one is the author talking about? If you are lucky, you are conscious that there are multiple possibilities. If you are unlucky, you flip to Example 2.4.7 but the author meant Theorem 2.4.7, and you go a bit mad trying to make Example 2.4.7 logically relevant to the reference.

PreTeXt makes it easy to avoid this particular annoyance, because cross references can identify their "type", especially through the use of the global "autonaming" feature. But even when the text explicitly says "and according to Theorem 2.4.7...", with humans being human, some readers will still focus on the "2.4.7" and begin a search for that number rather than the theorem with that number. With common counters, once 2.4.7 is a Theorem, there will never be an Example 2.4.7.

One counterargument is that it feels "wrong" to allow for an Example 2.4.9 and an Example 2.4.11 with no Example 2.4.10 existing in between. This violates our instinct to categorize and order objects. And perhaps the "missing" Example 2.4.10 will indeed cause some confusion to some readers. However, we suggest that such idealized views should be subservient to our goal of producing textbooks that provide a useful resource for the vast majority of our students.

We plan to allow figures and tables to use different counters, since they look obviously different, so you can quickly distinguish the previous, or next, item as you scan a page. Notice that it is their *distinctive appearance* that is the criteria for an independent counter. For example, numbers for displayed equations meet this criteria. They have their own counter, they are displayed distinctively when originlly formatted, and a cross-reference emphasizes their distinctive type through the use of parentheses (e.g., Equation (2.4.7)).

Also, as an author, recognize that there is a very flexible mechanism for making lists of objects that may be included in the `<backmatter>`. To continue the example here, you could make a list of all the theorems in the book, and a separate list of all the examples. Each list would be in the order of appearance, include the number (and a title if you provide one). In HTML output, each is a knowl which will quickly provide the content (independent of location), and also provides an "in-context" link to take you to the location for surrounding material. This useful feature requires very little additional effort, especially if you title your blocks as you author them.

Why do I have LaTeX errors? If you have errors when you run `pdflatex` or `xelatex`, that is more likely to be caused by a problem with your LaTeX installation than with PreTeXt. It is difficult to make legitimate PreTeXt that fails to compile; an exception being math markup inside a math element (`<m>`, `<me>`, and friends).

To check your LaTeX installation, run `pdflatex -version`. If it reports something older than "TeX Live 2017", then updating may help. If the trouble seems to be coming from a particular package, then check which version of the package is being used. For example, if the "tasks" package is causing a problem, run `kpsewhich tasks.sty` in the directory where you are compiling the LaTeX. If the package is in your personal texmf tree, that may be the problem.

If the PDF is created without errors, but something looks wrong in the PDF, then probably the PreTeXt source markup is wrong. Validate your source against the schema (see Section 5.5), and also carefully examine the HTML output. If that does not reveal the problem, seek expert help.

I have great output, so why does validation produce hundreds of errors? Success with a tool like `xsltproc`, in terms of no errors and great-looking output, does not mean your source is correct. XSLT

processing can be forgiving, and many invalid constructions just work, and look great. But that is no guaranteee this situation will continue, or the same happy accidents occur for a conversion to a different output format.

We do raise some errors during processing with `xsltproc`, but error-checking your input is a job for a validator, so in theory we should not ever produce any errors during a conversion. So strive for having 100% valid PreTeXt source, not simply great-looking results. See Chapter 5 and Appendix F.

Why do I have an "extra" period at the end of a title? Author your titles *without* punctuation at the end meant for spacing or separation (period, colon, semi-colon, hyphen). Do include punctuation which imparts meaning (question-mark, exclamation-point). PreTeXt will then add separation (a period, or spacing), as needed, in all the places where it is required. But PreTeXt will respect your question-marks and exclamation-points.

If you think you have an additional punctuation character that conveys meaning at the end of a title, please bring it to our attention.

Appendix C

Best Practices

Appendix D

Conversion from LaTeX

As part of the UTMOST project, we offer a service to help convert existing textbooks from LaTeX to PreTeXt. The service is free if you are planning to release your book with an open license. The conversion will only be 95 percent correct, but that means it will take you 20 times less effort than converting it yourself.

Before converting your book, familiarize yourself with PreTeXt to the point of being able to compile the sample article and sample book into HTML and PDF. Check that you can edit the source files and the resulting output files behave as expected. That way, you will be on familiar ground when you finish the last 5 percent of the conversion.

The actual conversion will be done by David Farmer from the American Institute of Mathematics. The first hurdle is to get your LaTeX files to David. The preferred method is:

1. Put your *entire* LaTeX source into a GitHub repository. Do not edit or restructure in any way, nor provide just a subset for "testing," since this only complicates the process. You do not need to include the image files.

2. If the repository is private, make David a collaborator so he can access it. (GitHub username: `davidfarmer`.)

3. Email `farmer@aimath.org` with your request for conversion, including the URL of your repository and a brief description of your project.

4. When the first draft of the conversion is ready, you will receive a pull request from David via GitHub. Go ahead and accept it, and now your repository will have new files that are the PreTeXt source.

5. Spend some time to review the files carefully, looking for consistent mis-interpretations of your intent. Convert to HTML and PDF and see how they look. There may be some back-and-forth with you explaining what your LaTeX was trying to do, and David improving the conversion to PreTeXt.

6. At some point it will be up to you to take ownership of the PreTeXt source and finish the conversion.

This service is for authors who wish to consider having PreTeXt as the "official" source of their textbook. It is not feasible to maintain LaTeX source and expect to have all of the features of PreTeXt.

Appendix E

(*) Text Editors

This appendix has information about using various text editors efficiently with MathBook XML source. The choice of an editor that suits you is a big part of being a productive author. Despite not being open source, we are partial to Sublime Text, due to its unlimited trial period, reasonable licensing (cost and terms), range of features, and cross-platform support. So we lead with Sublime Text, but also include Emacs and XML Copy Editor. A summary table of schema-aware editors can be found at Section 5.7.

E.1 Sublime Text

Dave Rosoff

Sublime Text is a fast cross-platform editor with thousands of user-contributed packages implemented in its Python API. It is not free or open-source, although most of the user-contributed packages are both. Development is active as of June 2016.

Here, we outline several of the most important Sublime Text features that will help you to minimize your typing overhead and work more efficiently with your MathBook XML project. We also introduce the MBXTools package designed to help MathBook XML authors work more efficiently.

Sublime Text 2 and 3 are both available for an unlimited evaluation period, but a licence must be purchased for continued use. I have found the additional features of Sublime Text 3 to be well worth the cost of the license.

E.1.1 Settings

Sublime Text settings are stored and managed in a collection of JSON files as key-value pairs, in files that have a `.sublime-settings` extension. You change the settings by visiting these files and editing the values away from their defaults.

To edit your Sublime Text settings, you can use the Preferences/Settings — User menu (Sublime Text/Preferences... on OS X). Make sure that when you go to edit Settings, you always choose the User option. Changes to Default settings files will be overwritten when Sublime Text updates. It is recommended to use the Default files to see what settings are available to change. There are a lot, and not all are documented.

All Sublime Text users should be aware that a particular view (buffer) may receive settings in several different ways, e.g., from global default settings, from global OS-specific settings, from package-provided settings, from user-provided settings, and so on.

Key bindings are also stored in files with a similar format. There are only so many keyboard shortcuts available, although Sublime Text does support multistep shortcuts like Emacs. If you find that you wish to reassign shortcuts, this is certainly possible through the Preferences/Key Bindings — User menu (Sublime Text/Preferences... on OS X).

E.1.2 Package Control

Sublime Text's Python API exposes a lot of the Sublime Text internals to plugin and package authors. Packages extend Sublime Text's functionality, much like Emacs major modes. A package usually consists of some Python scripts that define Sublime Text events and actions, some text-based configuration files (XML/JSON/YAML files defining language syntax, symbol recognition, custom snippet insertion triggers and contexts, keybindings for new and old commands, etc.), and perhaps some other stuff too. These typically get bundled into a `.zip` archive that is disguised with the unusual extension `.sublime-package`. These archives live in the Packages directory, accessible via the Preferences menu (the Sublime Text/Preferences menu on OS X). Sublime Text monitors the `Packages` directory for changes and reloads all affected plugins on the fly.

The first thing you should do after installing Sublime Text is install the Package Control package. This package manager operates within Sublime Text to automatically fetch updates for packages you have installed (unless you disable this feature). You can also list currently installed packages, find new packages to investigate, remove packages, etc.

Thousands of user-contributed packages are available for easy installation via Package Control. It is possible to maintain packages by hand, since most package authors publish via GitHub, but Package Control is the universally recommended method of obtaining, managing, and removing packages for your installation.

1. Visit the Package Control download site.
2. Find the Sublime Text console command (make sure the correct version of Sublime Text is selected) and copy it to the clipboard.
3. Open the Sublime Text console (`Ctrl-`` `) and paste the command into the window that appears, then press `Enter`.

Having installed Package Control, you can use the command palette to deploy its commands, such as Install Package, List Packages, and Remove Package. See the documentation for more. A few packages that are especially useful are recommended throughout this section, and summarized in Subsection E.1.9.

E.1.3 (*) Keyboard Shortcuts

To be written.

E.1.4 Project Management

Like many modern editors, Sublime Text has good project management features. These allow files that are part of a larger project to work together. For example, Sublime's Goto Anything command allows quick access to any file in a project. The Find in Project command permits users to search and replace (with or without regular expressions) across an entire project. Matches are displayed in a text buffer and double-clicking opens the relevant file at the appropriate position.

The sidebar provides a convenient view of all of the files and directories in a project—or, if you like, a filtered view, where files of your choice are excluded. The MBXTools package (Subsection E.1.7) also makes some use of project-specific settings in order to provide some of its functionality.

E.1.4.1 The Open Folder Command

The easiest way to make use of the project management functionality is to store related files in a single directory and its subdirectories. If you then use the File/Open Folder... command, the entire directory is opened and all its subdirectories and files are shown in the sidebar. You can toggle the sidebar with either the command palette or directly with `Ctrl+K, Ctrl+B` (`Cmd+K, Cmd+B` on OS X).

By making use of this command you are already using project management, even if you never save your project. Sublime Text always has an implicit project open if you don't open an explicit one. This is good enough for many users a lot of the time, since it provides the most useful feature (Find/Find in Project). The Goto/Go To Symbol in Project command is also useful, but not fully implemented in MBXTools (Subsection E.1.7). Some of the benefits of explicit project management are outlined below.

E.1.4.2 Explicit Projects

To save your project explicitly, use the Project menu to choose Save As Project... and choose an appropriate name and location. For a MathBook XML project, this would probably be the same name and location as the document root file. Use the Project menu commands to open and close your project.

There are a few benefits to using an explicit project to group files.

- You can group together files and folders in different parts of the filesystem, instead of being restricted to subtrees.
- You can have project-specific settings that are different from Sublime Text's defaults and different from your user preferences ((((subsection-settings)))).
- Sublime's project workspaces will remember which files you had open when you last closed the project, and at which positions.
- If you get very fancy, you can have multiple workspaces for the same project, with different filters and views for different purposes.
- It is fine to include `.sublime-project` files in Git repositories, but `.sublime-workspace` files should ***never*** be so included (according to the Sublime Text documentation).

E.1.4.3 Using the Sidebar

The project sidebar allows you to view the entire directory tree (rooted at the folder you opened with the Open Folder command), or, if you've opened an explicit project as described above, all of its files and folders. You can use the sidebar to copy, move, rename, delete, and duplicate files, for example, as well as opening them.

The package SideBarEnhancements is highly recommended (install via Package Control). It makes the sidebar much more useful.

An alternative to the sidebar that Emacs users especially will find helpful is the dired package. The link is to a git repository since the package is no longer available from Package Control. This package allows you to browse the directory tree in a Sublime Text buffer. You can rename and move files within it—using all your favorite Sublime commands, including multiple selections (Subsection E.1.5). You might also try the SublimeFileBrowser package, which is actively maintained, available in Package Control, and seems to provide similar functionality.

E.1.5 Multiple selections

Multiple selections are the single most useful and irreplaceable feature of Sublime Text, the one that will keep you coming back. From the documentation:

> Any praise about multiple selections is an understatement.

The base functionality of multiple selections is simple. Hold down the `Ctrl` key (`Cmd` on OS X), and click somewhere in the open view to get a second cursor. Continue to add more cursors. All of them will behave together when you type: text will be inserted, most snippets or other text commands function as usual, etc. Even mouse commands work in an intuitive way with multiple selections.

It is hard to explain exactly what makes multiple selections so powerful. You just have to try it for yourself. Here is a typical example. In a structured document, many bits of text occur quite frequently—element and attribute names, for example. You may want to update several occurrences of a fragment at once—making several identical changes. Sublime's Quick Add Next command (`Ctrl+D`/`Cmd+D`) makes this a snap.

1. Place the caret somewhere in the word you'd like to modify.
2. Use Quick Add Next to expand your (empty) selection to the current word.
3. Use Quick Add Next again to add the next instance to the selection, which will then typically be disconnected.

4. Continue to Quick Add Next as many times as you like. Use Quick Skip Next (`Ctrl+K, Ctrl+D`/`Cmd+K, Cmd+D`) to jump over instances you would like to leave alone. If you go too far and select in error, hit `Ctrl+U/Cmd+U` to undo.

5. Make your modification, only one time.

Another example that occurs frequently when authoring XML is when you use the Wrap with Tag snippet (`Alt+Shift+W`/`Ctrl+Shift+W`). This snippet wraps the selection(s) in a `<p>` tag, with the tag name highlighted in both the start and end tags. If the `p` element is not what you wanted, just type. Both tags are replaced. This is a huge benefit to the XML author that makes essential use of multiple selections, even though you are barely aware of this as you use the feature.

Column selection allows you to select a rectangular area of a file. This is unbelievably useful when editing a structured document. There are lots of ways to do it (see the Sublime Text documentation for a mostly exhaustive list), but the most frequently used is to hold down `Shift` while clicking and dragging with the right mouse button (on OS X, hold down `Option` while dragging with the right mouse button). See the documentation for keyboard-based shortcuts.

Column selection becomes even more useful when used in combination with the keyboard shortcuts for moving and selecting, such as `Ctrl+Shift+Right` (select to end of word) and `Shift+End` (select to end of line).

Yet another example of the appallingly great utility of multiple selection comes when copying and pasting from a different file format. Suppose you have copied some lines of text and wish each such line to become a list item in your MathBook XML source.

1. Use column selection, as described above, to select each line individually.

2. Use Wrap with Tag to wrap each of the selected lines with matched begin/end `<li>` tags, all at once.

3. Now you have to select the lines again, to wrap them with matched begin/end `<p>` tags. First, hit `Shift+End` to select to end of line.

4. If your lines are wrapped, you may need to hit `Shift+End` again to get to the end of the wrapped lines.

5. Now you've selected too far: the `</li>` are selected as well. Hold down `Ctrl+Shift` and hit the left arrow twice (unselect by word). (After a little practice, steps like this seem automatic.)

6. Use Wrap with Tag to wrap each of the selected lines with matched begin/end `<p>` tags, all at once.

This does take a little mouse-work, but the keystroke savings can be considerable. (The Emmet package, described in Subsection E.1.6, provides an even quicker way to do this task and much more complicated ones.)

There are so many incredibly handy ways to use multiple selections that we will forgo any further examples to leave the reader the pleasure of discovering her own favorites. One particularly helpful package is Text Pastry, which provides some autonumbering and text insertion commands that work nicely with multiple selections. There are also a handful of packages that extend multiple selection functionality, such as PowerCursors and MultiEditUtils. PowerCursors allows you to add cursors and manipulate them without using the mouse. MultiEditUtils provides additional text processing commands designed to work with multiple selections.

E.1.6 Emmet

Emmet is the most downloaded plugin for Sublime Text (1.82 million installs via Package Control). It is mostly used by HTML and CSS authors and provides a lot of functionality for them. It is also useful for writing XML, as we see below. The main benefits of working with Emmet are ease of tag creation, manipulation, and removal.

Emmet by default overrides Sublime's binding for the `Tab` key, endowing it with new behavior (the command Expand Abbreviation). This new behavior is to create a matching XML tag pair for whatever word is to the left of the caret, or with whatever words are selected. For example, if you were to type "ol" and press the `Tab` key, the resulting text would be

```
<ol></ol>
```

with the caret positioned between the two newly created tags. Pressing `Tab` a further time moves the caret to the right of the end tag.

Emmet will produce any word it does not recognize into a matched tag pair when the Expand Abbreviation command is run. Some XML elements are empty, though. Within a matched tag pair, the command Split/Join Tag (`Ctrl+Shift+`/Cmd+Shift+`) will contract it into an empty tag, removing any text between the existing begin and end tags. (If the caret is *inside* a tag for an empty element, this command replaces the empty element with a matching begin/end tag pair.)

The default behavior (creating tag pairs whenever `Tab` is pressed) interferes with Sublime Text's usual Tab-completion, which may be undesirable. It may be disabled by setting

```
"disabled_keymap_actions": "expand_abbreviation_by_tab"
```

in the Preferences/Package Settings/Emmet/Settings — User file. The functionality of Expand Abbreviation will still be available through `Ctrl+E`.

For a more involved example of abbreviations, suppose you have pasted the items of an ordered list. Now you need to structure it with `ol`, `li`, and so on.

```
Lists are often good.
You can provide list items with <c>@xml:id</c>.
You probably don't want to number them, though.
```

The desired output is:

```
<ol>
    <li xml:id="item1">Lists are often good.</li>
    <li xml:id="item2">You can provide list items with <c>@xml:id</c>.</li>
    <li xml:id="item3">You probably don't want to number them, though.</li>
</ol>
```

Using Emmet, one produces it by executing the Wrap as you Type command (`Ctrl+Shift+G`/`Ctrl+W`) and entering the following expression in the minibuffer.

```
ol>li[xml:id=item$]*>p
```

The `>` symbol denotes a child element, the square brackets (with or without assignment) denote an attribute list, the `$` provides the line-based numbering, and the `*` specifies wrapping each selected line with the indicated subtree (so each line is wrapped with `<li><p>`, instead of the entire selection).

Emmet can produce a large hierarchy of nested XML tags at various levels using this abbreviation syntax. For example, suppose you know that you will need to produce a tag structure of the following form.

```
<section xml:id="">
    <introduction>
        <p></p>
    </introduction>
    <subsection xml:id="">
        <p></p>
        <p></p>
        <figure></figure>
        <p></p>
        <ol>
            <li></li>
            <li></li>
            <li></li>
        </ol>
    </subsection>
    <conclusion>
```

```
        <p></p>
    </conclusion>
</section>
```

Admittedly, this is a bit much, but it makes the point. The Emmet "abbreviation" for this structure is:

```
section[xml:id]>introduction>p^(subsection[xml:id]>p*2+figure+p+ol>li*3)^^conclusion>p
```

Upon typing this string and placing the caret to the right of it, hit `Ctrl+E` (or `Tab`, if you didn't disable the Emmet default). The entire tree structure is created immediately, with tab stops for the missing attribute values and for each matching begin/end pair.

The Expand Abbreviation As You Type command allows you to tweak such abbreviations interactively. Hit `Ctrl+Alt+Enter` and type the expression above into the minibuffer at the bottom of the window, watching the tree appear as you type.

Emmet is a very powerful package that can do much more than is outlined here. However, it is by default mostly adapted to writing CSS and HTML. Customizing it to work more directly with MathBook XML is an ongoing project. You can discover more about Emmet by examining the Emmet documentation or poking around in the Settings and Keymap files.

E.1.7 MBXTools—a Sublime Text package for MathBook XML

MBXTools is a Sublime Text package designed to assist authors using MathBook XML. It is very experimental and may behave unexpectedly.

The package owes its inspiration and much of its code to the excellent LaTeXTools package. Please let the author know of any bugs you find or any features you would like to see included in MBXTools by creating a GitHub issue.

E.1.7.1 Installation

via Package Control. It is recommended to install MBXTools via Package Control. If you have not installed Package Control yet, you should do that first (and restart Sublime Text afterward).

After Package Control is installed, use the `Install Package` command to search for the MBXTools package, and select it from the Quick Panel to install. This method of installation allows Package Control to automatically update your installation and show you appropriate release notes.

via git. You may also install MBXTools via `git`. Change directories into your `Packages` folder. To find the `Packages` folder, select Browse Packages from the Preferences menu (from the Sublime Text 3 menu on OS X). Make sure you are in the `Packages` folder and *not* `Packages/User`.

Then, run

```
git clone https://github.com/daverosoff/MBXTools.git
```

and restart Sublime Text (probably not necessary).

E.1.7.2 Usage

You can activate the package features by enabling the MathBook XML syntax. The syntax definition looks for `.mbx` file extensions, which most of us do not use (yet?). If your MathBook XML files end with `.xml`, you will either need to add a comment to the first line of each file (after the XML declaration):

```
<!-- MBX -->
```

or you will need to enable the syntax manually using the command palette. To enable it manually, open a MathBook XML file and press `Ctrl+Shift+P` (`Cmd+Shift+P` on OS X) and type `mbx`. Select "Set Syntax: MathBook XML" from the list of options.

You should see the text "MathBook XML" in the lower right corner if you have the status bar visible (command palette: Toggle Status Bar).

There are only a few features implemented so far.

1. If you have some sectioning in your MBX file, hit `Ctrl+R` (`Cmd+R` on OS X) to run the Go To Symbol command. You should see a panel showing all the divisions' `@xml:id` names.
2. If you have been using `@xml:id` to label your stuff, try typing `<xref ref="` (the beginning of a cross-reference). Sublime Text should show you a panel containing all `@xml:id` values along with the elements they go with. Choose one to insert it at the caret and close the `xref` tag. Alternatively, type `ref` and hit `Tab` to activate the `xref` snippet. Then hit `Ctrl+l` followed by `x` or `Ctrl+l` followed by `Ctrl+Space` to bring up the completions menu. There are several variants of the `ref` snippet, namely `refa`, `refp`, and `refpa`.
3. Type `chp`, `sec`, `ssec`, or `sssec` and hit `Tab` to activate the division snippets. A blank `title` element is provided and the cursor positioned within it. As you type, the `@xml:id` field for the division is filled with similar text mirroring the title you are entering.

E.1.7.3 Known issues

1. When manually adding an `xref` (not using the snippets or autocomplete), you will frequently see a spurious "Unrecognized format" error.
2. The `ref` snippet does not bring up the quick panel. Should it?
3. Recursive search through included files for labels is not yet implemented.

   This will only work for `xref` completion, not Go To Symbol.
4. Nothing has been tested on OS X or Linux.

E.1.8 (*) Sublime Linter

To be written.

E.1.9 Recommended Packages

1. Package Control
2. Emmet
3. SideBarEnhancements
4. PowerCursors
5. MultiEditUtils
6. Text Pastry
7. Git or SublimeGit
8. SublimeLinter
9. MBXTools

E.2 emacs

Jason Underdown reports on 2016-05-12 that emacs' nXML mode (`www.gnu.org/software/emacs/manual/html_mono/nxml-mo`
works well with a RELAX-NG schema. While we work on building a hand-crafted RELAX-NG schema, you can use the trang (`www.thaiopensource.com/relaxng/trang.html`) tool to convert the PreTeXt DTD to a RELAX-NG schema.

> You simply put your cursor at any point in the document, start a new tag with < and then call the `completion-at-point` function (I bound it to the key-chord: `C-<return>`) to get a list of possible completions. Or you can start typing a few characters to narrow the list of possibilities. It will also let you know if the element you are trying to insert is invalid.
>
> —Jason Underdown

E.3 XML Copy Editor

Michael Doob reports on 2017-02-03 that XML Copy Editor (`xml-copy-editor.sourceforge.net`) works well, in particular on Windows. This is an open source program, for Windows and a variety of popular Linux distributions, that supports both DTD and RELAX-NG schemas. It is less of a general programmer's editor and more like dedicated tools for working strictly with XML documents.

E.4 (*) vi, vim

Contributions welcome.

Appendix F

Schema Tools

This appendix has technical information about tools that work with the RELAX-NG schema and Schematron rules. See Chapter 5 for a more general overview.

F.1 Jing and Trang

These tools come from James Clark, an author of the RELAX-NG syntax. `trang` is a converter between different formats for schemas. An author should not ever need it. Though a PreTeXt developer might find it useful, and it is a by-product of the `jing-trang` install described below.

`jing` is our recommendation for RELAX-NG validation by authors, and works very well.

`jing` and `trang` are packaged (separately) for Debian/Ubuntu Linux. Reports of other similarly easy installations for other operating systems, to be included here, are especially welcome. We have employed the Debian/Ubuntu versions and also versions built from source (next).

If you cannot install this tool easily as part of your system, you can still follow the notes below to build from source. Many authors have done this successfully. Installation notes for `jing` and `trang` follow.

Clone Clone the `git` repository at github.com/relaxng/jing-trang with the command

```
git clone https://github.com/relaxng/jing-trang
```

which assumes you have a command-line version of `git` installed. You will end up with a new `jing-trang` directory, which you will certainly want *outside* of your PreTeXt files and *outside* of your project files.

`readme.md` Follow the instructions in the `readme.md` found at the top level of the `jing-trang` distribution, observing the following notes keyed to the four steps. These helpful notes come courtesy of the experiences of Jahrme Risner, Mitch Keller, Bruce Yoshiwara, Ken Levasseur, and Jessica Sklar on a variety of operating systems.

1. It is necesssary to have a developer's version of `java` on your machine. Try `which java` to see if one is already available. Any output here might help in the next step. References here to the JDK is the Java Development Kit.
2. You will need to have the `JAVA_HOME` environment variable set correctly. You can try `echo ${JAVA_HOME}` in a console to see if it produces anything sensible and/or consistent with Step 1. Ken Levasseur notes that on OSX you can go

   ```
   echo $(/usr/libexec/java_home)
   ```

   and the output is what you set to the `JAVA_HOME` variable.

   On Windows you may need to set Environmental Variables in the Windows System Properties GUI, both here and in Step 4.

3. Setting your working directory to the root directory of the `jing-trang` distribution should not cause any particular difficulties.
4. You may need to install the `ANT` tool, almost certainly on Windows. Again, on Windows you may need to set an `ANT_HOME` environment variable. On Linux, this may be all set for you already as a system tool.

   The README suggests changing slashes on Windows. But you may already be using a shell that gives Unix-like behavior. So try both directions, if necessary. Also, the README suggests running `./ant test` (or `.\ant test`). Doing this on Windows may yield a `BUILD FAILED` message, but `jing` may still work.

Results Change into the `build` subdirectory. You may have more files here, but the two you really want are `jing.jar` and `trang.jar`. So if you have these, you are in good shape.

If you have modularized your source files (Section 4.2) then you need one more library. Look in the top-level `lib` directory (a peer of `build`) for `xercesImpl.jar`. You have two and a half choices now.

- Copy `lib/xercesImpl.jar` into `build`.
- Or make a "symbolic link" to the third JAR archive. Be sure you are in `build` and go

  ```
  ln -s ../lib/xercesImpl.jar
  ```

  which will just make your operating system think this third archive is in the `build` directory.
- A third option would be to adjust/augment some classpath information below and send us a report of success. We have not tested this approach.

Now you are ready to use `jing` to validate a PreTeXt document. Note that the commands below require the XML version of the schema, which is the non-compact version. Also, these are the commands if you build `jing-trang` from source. If you install a system-supplied version, then consult the `man` pages, or similar, for syntax which is likely much easier and direct.

For a document contained in a single file, run (with suitable working directory and path prefixes).

```
java -jar jing-trang/build/jing.jar pretext.rng my-book.xml
```

For a document modularized across several files using the `xinclude` mechanism, issue as one single command line (again, with suitable working directory and path prefixes). This presumes you have moved or symlinked the `lib/xercesImpl.jar` file into the `build` directory. Do not leave any spaces around the equals-sign, we have split that line there for readability, so the `-D` option should not have any spaces in its argument.

```
java -classpath jing-trang/build
-Dorg.apache.xerces.xni.parser.XMLParserConfiguration=
    org.apache.xerces.parsers.XIncludeParserConfiguration
-jar jing-trang/build/jing.jar pretext.rng my-book.xml
```

It may go without saying that scripting this task will make you more likely to do it as often as is necessary and you will save yourself much time, and a little frustration, in the process.

Jahrme Risner provides the following setup he uses to make it very convenient to regularly validate his sources. This is a "shell script", which a Linux user might add to their `~/.bashrc` file. Note that we have again split the line with the `-D` option at the equals-sign, without a line-continuation character. Do not do that in your version.

```
ptx-check() {
  java\
    -classpath ~/GitHub/jing-trang/build\
    -Dorg.apache.xerces.xni.parser.XMLParserConfiguration=
        org.apache.xerces.parsers.XIncludeParserConfiguration\
    -jar ~/GitHub/jing-trang/build/jing.jar\
    ~/GitHub/mathbook/Schema/pretext.rng "$1"
}
```

Then he can simply go

```
~ $ ptx-check my-book.xml
```

at anytime. Note that you might have to provide or adjust some of the paths above for your situation. And there are other ways to script tasks like this.

F.2 Schematron

The second step of validation is checking the Schematron rules. Fortunately, there is nothing to install. Use is just like the conversions you have already been doing. In the PreTeXt distribution, there are three files related to Schematron in the `schema` directory. Two of them are meant for developer use, or for the curious. The file you are interested in is `pretext-schematron.xsl`. This is a stylesheet, unique to PreTeXt, which will carefully analyze your source to find any exceptions that the RELAX-NG schema was not designed to catch. You use the stylesheet like any other,

```
xsltproc schema/pretext-schematron.xsl ~/books/aota/animals.xml
```

with suitably adjusted paths, and be sure to provide the `-xinclude` switch if your source is modularized across multiple files (Section 4.2). No news is good news, but each exception found should provide enough explanation for you to locate, and correct, the problem. These messages are under PreTeXt's control, so please report any that are not helpful enough. That's it—easy.

Appendix G

Revision Control: git

Authoring a textbook without revision control is like driving without a seat belt. Sooner or later, you will wish you had used it. git is a popular program for revision control for software projects, and works quite well with PreTeXt, though not perfectly. Notes here are designed to help. For more on git itself, in the context of authoring a book, see Git for Authors, by Robert Beezer and David Farmer at `mathbook.pugetsound.edu/gfa/html`.

Word Wrap. git is designed for code, where a newline often expresses the end of a statement. In PreTeXt, it might make sense to author an entire (long) paragraph without any newlines. If so, a line-oriented file diff is not so useful. Fortunately, git has a flag, `--word-diff`, which does an excellent job of displaying small edits precisely.

Messages for Commits and Merges. When you make a commit or merge, you can supply a message at the command line with the `-m` argument. Otherwise you get thrown into an editor, with the default being `vi`, which can be hard to get out of if you have not used it before. Better, as Joe Fields suggests, is to tell git which editor you want to use. To set `pico` as the default editor, the one-time command-line incantation would be:

```
git config  --global core.editor "pico"
```

You can also directly edit the configuration file at `~/.gitconfig`. More suggestions can be found on this thread on StackOverflow at `stackoverflow.com/questions/2596805`.

Appendix H

Windows Installation Notes

Dave Rosoff

This appendix explains the best known way to install the PreTeXt toolchain in a Windows environment, rather than a Unix-flavored operating system (such as Linux and Apple's OS X). It has been tested on Windows 7, 8, and 10. We assume that none of the listed tools or equivalents have been previously installed. That may complicate matters. This is especially true if you use Cygwin, or if you have already installed Python on your machine. PreTeXt compatibility with existing Python installations is addressed elsewhere in this document ((((python-mbx-compatibility)))).

If you have Windows 10, be sure to read about WSL in Appendix I, which could be a whole lot easier to setup and maintain.

H.1 Setup

In this section, we do some initial setup, establish notation, and issue warnings. Some of the steps in this process are dangerous. Typos could lead to an unstable system, or possibly even to unrecoverable system errors. Double-check everything.

H.1.1 Notation

Strings enclosed in `<angle brackets>` are variables whose values you should substitute in typed expressions. `<username>`, for example, should be replaced with your Windows username (e.g., mine is `drosoff`). Throughout this installation process it is very important to pay attention to the direction of slashes `/` and backslashes `\`.

H.1.2 Initial Windows setup

It is easier to see what is happening if your Windows file browser is not set up to hide file extensions from you. Disable the "hide file extensions" behavior before proceeding. In Windows 7/8, this can be done through the Control Panel. In Windows 10, there is a checkbox somewhere in the ribbon for it.

- On Windows 7 or 8:
  1. Open the Start Menu and type "Control Panel". Select the `Control Panel` entry from the popup list.
  2. Type "Folder Options" into the search box in the Control Panel window. Select `Folder Options` when it appears.

3. Select the View tab.
4. Uncheck the box for "Hide extensions for known file types".
5. Click OK until there are no more OKs to click.

- On Windows 10:

1. Open the Start Menu (icon shaped like a Window at the bottom left, typically). Select the `File Explorer` option, right above `Settings`, from the popup list.
2. Click on this, and then select `View` from the "ribbon lists" of options along the top.
3. After clicking this, on the right there should be a check box for "File Name Extensions". Click this box; that should do it.

List H.1.1: Initial Windows Setup

H.1.3 A word on path names

An appallingly large fraction of the difficulties of using GNU/Linux-based utilities with Windows come from the differences in formatting path names. Windows path names begin with a drive letter (usually "C") and a colon. Like all path names, they describe a path in a rooted tree. The root directory (folder) in Windows is called `\`, a backslash. Note the direction carefully. Children of the root node are either subdirectories or files in the root directory (leaves). The path to my downloads folder is:

```
C:\Users\drosoff\Downloads\
```

The trailing backslash is often unnecessary, but it is an easy way to see immediately whether a path name refers to a file or to a directory. Windows path names are not case-sensitive.

Linux/Mac OS X path names are quite similar, but lack drive letters, start with an explicit reference to the root, use forward slashes, and are case-sensitive (more or less so, in Mac's case). A path to a typical Linux user's download folder might be

```
/home/typical.username/Downloads
```

Again, Linux pathnames are case-sensitive and Mac OS X pathnames are typically 'case-preserving'. The Git Bash shell for Windows is an emulation of a Linux environment, and the utilities within it expect path names that follow Linux conventions. So we conform to this expectation as follows.

1. Remove the colon, but keep the drive letter.
2. All backslashes `\` become slashes `/`.
3. Add an initial slash preceding the drive letter.

The path name to my Windows download folder becomes

```
/c/users/drosoff/Downloads
```

Even though Git Bash is pretending to be a Linux shell, path names are still the underlying Windows path names, and therefore are not case sensitive. You can verify this using tab-completion.

Path names that begin with the drive letter (Windows) or the root `/` (Linux/Mac OS X) are called **absolute path names**. Their referents do not depend on the location from which the path name is invoked. **Relative path names**, on the other hand, begin in the so-called **current working directory**. A relative pathname might look something like this:

```
../../examples/sample-article/sample-article.xml
```

The symbol .. is a shortcut for the parent of the current directory. Thus, the relative path name above means from where we are, ascend two levels, then descend into the `examples` and `sample-article` subdirectories, and find the file `sample-article.xml`.

Path names that contain spaces are *evil*, and should be avoided in many cases. Unfortunately, all Windows default program installation locations contain at least one space (directory name "Program Files"). This does not appear to cause problems, except when installing ImageMagick (Section H.5). To be extra careful, you could always choose an installation location that is free of space characters.

H.1.4 Do I have 64-bit Windows?

Find out on Windows 7:

1. Open the start menu and type "Computer". Right-click the `Computer` item in the popup menu.
2. Select `Properties` from the drop-down menu.
3. Read in the right-hand side of the pane to find the "System" heading.
4. From the "System type" entry, read whether you have a 32- or a 64-bit OS.

Now you are ready to begin installing the various pieces of the PreTeXt puzzle. The first one, `xsltproc` (Section H.2), is the most annoying. You might want to go get a snack or another cup of coffee.

H.2 Installing xsltproc

H.2.1 xsltproc binaries

This is the most annoying part of the installation. Obtain four zip archives from Igor Zlatkovic's FTP site that hosts the most recent Libxml binaries for Windows. At the time of this writing, the 64-bit binaries were considered experimental. I have had no trouble using the 32-bit binaries on my 64-bit Windows 7 system, so I suggest that all MBX users download the most recent 32-bit version of the following libraries:

1. `iconv` (filename something like `iconv-1.9.2.win32.zip`)
2. `libxml2` (filename something like `libxml2-2.7.8.win32.zip`)
3. `libxslt` (filename something like `libxslt-1.1.26.win32.zip`)
4. `zlib` (filename something like `zlib-1.2.5.win32.zip`)

List H.2.1: `xsltproc` Zip Files

We only need a handful of files from these archives. So the simplest thing is to leave them in your Downloads folder and grab what we need. Create a new folder `C:\xsltproc` (it can be anywhere, as long as it's a new location). We'll call this location `<xsltproc>` in case you named your folder something different.

Extract the following files from the four zip archives you downloaded above into `<xsltproc>`:

1. From `iconv-*.win32.zip`:
   (a) `iconv-*.win32\bin\iconv.dll`

2. From `libxml2-*.win32.zip`:
   (a) `libxml2-*.win32\bin\libxml2.dll`
   (b) `libxml2-*.win32\bin\xmllint.exe`
3. From `libxslt-*.win32.zip`:
   (a) `libxslt-*.win32\bin\libexslt.dll`
   (b) `libxslt-*.win32\bin\libxslt.dll`
   (c) `libxslt-*.win32\bin\xsltproc.exe`
4. From `zlib-*.win32.zip`:
   (a) `zlib-*.win32\bin\zlib1.dll`

List H.2.2: `xsltproc` Extracted Zip Files

H.2.2 Change PATH environment variable

Note: if you prefer not to meddle with this, it can be avoided. Now, we need to make sure your system can find these files when we need them.

1. Open the Start menu and start typing "Edit the system environment variables". Select this option when it becomes visible.
2. Click the Environment Variables button near the bottom of the dialog.
3. In the bottom part of the dialog labeled "System environment variables", look for a variable named `PATH`. You may need to scroll.
   (a) If you do not find one, create it using the New... button. Make sure to use all capital letters. (This really shouldn't happen. Make sure you are editing the *system* environment variables, not the *user* environment variables.)
   (b) If you do find the `PATH` variable, select it and click the Edit... button.
4. Regardless of which of steps 1 and 2 you followed, now you should see a dialog with two text fields. Your variable name should be `PATH`.
5. If you created this variable, populate the second field with the full path name of `<xsltproc>`, the location where you put the seven files from Igor's zip archives. For me this looks like `C:\xsltproc`.
6. If you are editing the `PATH` variable, place the cursor in the existing value and press the End key, so that the cursor moves to the back of the line. The `PATH` string is a `;`-delimited list of full path names, so append the string `<xsltproc>;` (note the semicolon) to the existing value. If you named <xsltproc> as we suggested above, then the last part of your `PATH` variable is now `C:\xsltproc;`.
7. Click OK to save changes.

List H.2.3: Path Environment Variable for `xsltproc`

Congratulations, you have successfully installed `xsltproc`.

Note that you have installed the `xmllint` utility as part of this procedure. This utility will allow some text editors to **lint** your PreTeXt files, that is, to automatically detect and highlight errors, and perhaps even to explain them.

H.3 Installing git

In this section we install the `git` version control system and some tools to interact with it, including a fairly full-featured emulation of the `bash` command line shell. I strongly recommend you use the Git Bash shell or another `bash` emulation, so that you can use Linux commands referenced elsewhere.

One feature in particular, `pdfcrop`, has not been made to work in the normal Windows `cmd` shell, although the rest of PreTeXt has. To generate images using the `mbx` script Section 9.9, you will need the Git Bash shell or something like it.

H.3.1 Steps to install git

1. Visit the official `git` download page (download starts automatically) and obtain the latest binary for your system.
2. Find the installer in your Download location and run it.
3. Choose whatever location you like for the `git` installation folder. I recommend you use the default.
4. At the "Adjusting your PATH environment" dialog, select either of the first two items. I recommend the second, "Add git executable to Windows PATH". This will allow you to use `git` from within other Windows programs, such as Sublime Text or other text editors, which can be extremely convenient. If you are apprehensive about adding `git` to the Windows PATH, select the first option. I do not recommend the third option.
5. Accept the default options for all the remaining prompts.

List H.3.1: git Installation

H.3.2 Changing the path with .bashrc

In Subsection H.2.2, we promised that you could avoid messing with the Windows environment variables. If you install something else later that wants to use `xsltproc`, then this might not be the best idea. But if you are only going to use it from within Git Bash, then this will work fine.

From the Git Bash command prompt, enter this line of text and hit Enter. Do not make any typos. You should substitute your value of `<xsltproc>` where indicated, but make sure to conform to the conventions at the end of Subsection H.1.3 regarding Windows path names in Git Bash. (I warned you this was going to be annoying.)

```
echo "export PATH=<xsltproc>:$PATH" >> ~/.bashrc
```

You may get a message from Git Bash the next time you run it about `.bash_profile`, which you may safely ignore.

Congratulations, you have successfully installed `git`.

H.4 Installing Anaconda

Anaconda is a well-regulated development environment for Python under Windows, and I recommend it for users who do not already have Python installed. The essential `mbx` script has recently been updated to support both Python 2 and Python 3. Therefore we make no recommendation about which Python version to choose.

If you already have a working Python installation, skip to Section H.5.

You have some choice with your Anaconda installation. It actually supports the installation of several independent Pythons side-by-side (since both 2.7 and 3.x are in active use, this is more reasonable than it seems). If you do not need Python for anything else or are simply a minimalist, Miniconda is also an option. Miniconda installs no packages, but these can be installed via the `conda` utility at the command line later.

1. Download either Miniconda, Python 2.7, or Python 3.5 from the Anaconda download page.
2. Run the installer and accept all the default suggestions.

Congratulations, you have successfully installed Python.

If you do not care about images, you can stop here. Much of the MBX functionality is already present. However, to use the `mbx` script to create SVG images from sources like PDF/PNG images, Sage, Asymptote, or TikZ, you need to install ImageMagick and Ghostscript using the directions in Section H.5 and Section H.6.

H.5 Installing ImageMagick

Visit the ImageMagick downloads page and grab a binary. If you have a 64-bit Windows installation (Subsection H.1.4), use the recommended version. If you have a 32-bit installation, find the version whose filename is obtained from that of the recommended version by substituting `x86` for `x64`. For example, if the recommended version's filename is:

```
ImageMagick-7.0.1-6-Q16-x64-dll.exe
```

then a good choice for a 32-bit Windows would be

```
ImageMagick-7.0.1-6-Q16-x86-dll.exe
```

1. Run the installer from your download location.
2. Accept the license agreement.
3. Choose a default installation location that has no spaces in its folder name. The default choice "Program Files" causes problems because of path name issues. I chose

   ```
   c:\ImageMagick-7.0.1-Q16\
   ```

   It matters because the ImageMagick utility `convert` is used by the `mbx` script to convert your images into different formats. The `mbx` script will have a lot of trouble with path names that contain spaces.
4. When confronted with "Select additional tasks", make sure that the boxes for "Add application directory to your system path" and "Install legacy utilities" are checked.
5. If you like, carry out the procedure to verify your installation.

List H.5.1: imagemagick Installation

Congratulations, you have successfully installed ImageMagick.

H.6 Installing Ghostscript

Visit the Ghostscript download area and download the most current binary for either 64-bit or 32-bit Windows (Subsection H.1.4). Run the installer. You can accept almost all the default options. As with ImageMagick (Section H.5), it is probably best to choose an installation location whose path name is free of spaces, such as `c:\gs`. We refer to this installation location as `<gs>` below.

H.6.1 Change PATH environment variable

The `pdfcrop` utility needs to be told which Ghostscript command to use. We need to add the file `gswin64c.exe` to the Windows PATH. This is similar to what is done above, in Subsection H.2.2.

1. Open the Start menu and start typing "Edit the system environment variables". Select this option when it becomes visible.
2. Click the Environment Variables button near the bottom of the dialog.
3. In the bottom part of the dialog labeled "System environment variables", look for a variable named `PATH`. You may need to scroll.
4. If you do find the `PATH` variable, select it and click the Edit... button.
5. You should see a dialog with two text fields. Your variable name should be `PATH`.
6. Place the cursor in the existing value and press the End key, so that the cursor moves to the back of the line. The `PATH` string is a `;`-delimited list of full path names, so append the string `<gs>;` (note the semicolon) to the existing value. If you named <gs> as we suggested above, then the last part of your `PATH` variable is now `C:\gs;`.
7. Click OK to save changes.

List H.6.1: Path Environment Variable for `ghostscript`

Congratulations, you have successfully installed Ghostscript.

H.7 Installing `pdf2svg`

The installation procedure uses `git`. Open Git Bash and change to your root directory:

```
cd /c
```

Clone the repository into `C:\pdf2svg`:

```
git clone https://github.com/jalios/pdf2svg-windows.git pdf2svg
```

H.7.1 Change PATH environment variable

We need to add the `pdf2svg` program to the Windows PATH. This is similar to what is done above, in Subsection H.2.2 and Subsection H.6.1.

1. Open the Start menu and start typing "Edit the system environment variables". Select this option when it becomes visible.
2. Click the Environment Variables button near the bottom of the dialog.
3. In the bottom part of the dialog labeled "System environment variables", look for a variable named `PATH`. You may need to scroll.
4. If you do find the `PATH` variable, select it and click the Edit... button.
5. You should see a dialog with two text fields. Your variable name should be `PATH`.
6. Place the cursor in the existing value and press the End key, so that the cursor moves to the back of the line. The `PATH` string is a `;`-delimited list of full path names, so append the string `C:\pdf2svg\dist-32bits;` or `C:\pdf2svg\dist-64bits;` (note the semicolon) to the existing value.
7. Click OK to save changes.

List H.7.1: Path Environment Variable for `pdf2svg`

Congratulations, you have successfully installed `pdf2svg`.

H.8 What's Missing

Development of a Windows-compatible `mbx` script (Chapter 9) is mostly complete. If you need help with `mbx`, contact Dave Rosoff. There are still a few use cases that haven't been tested, mostly those to do with Asymptote.

At present, it only seems to be possible to install Sage on Windows by way of the Windows Subsystem for Linux, available only in Windows 10.

If you find any problems or bugs, please let us know at the PreTeXt Support group in Google Groups, or email `drosoff` AT `collegeofidaho` DOT `edu`.

Appendix I

Windows Subsystem for Linux

With Windows 10 you can install the **Windows Subsystem for Linux** (WSL). This is basically Ubuntu Linux (one of the most popular versions of Linux) integrated into Windows 10 in a way that command-line Linux programs can be executed easily. News and announcements can be fouund at `https://msdn.microsoft.com/commandline/wsl/about` (`msdn.microsoft.com/commandline/wsl/about`). Michael Doob reports on 2017-06-02 that this works quite well for the programs necessary to author with PreTeXt, and provides the following help.

Installing WSL. If you have the "Anniversary Edition" of Windows 10 (later than August 2016), then installing WSL is not difficult. Just follow the (reasonably straightforward) instructions given by Microsoft at the address `https://msdn.microsoft.com/en-us/commandline/wsl/install_guide`.

Upon completion of the installation, you should

- be able to use the `bash` command from the `PowerShell` window,
- have your own WSL userid (distinct from Windows),
- have your own WSL password (distinct from Windows).

A little background about using about the command line.

- You type in commands (terminated by the Enter key) and the operating system responds. For example, if you type in `date`, the operating system responds with (what it considers to be) the date. Using the command line is an ongoing conversation between you and the operating system.
- The `sudo` command: when a command starts with `sudo`, the rest of the command is executed with administrative privileges. This is needed, for example, to install software or update the operating system. You must give your password when you run `sudo` (although you get a little window of time after the first usage when it is not necessary to do so).
- The `sudo apt-get update` command: this is used to resynchronize the local listing of installed packages with those in the official repository.
- The `sudo apt-get upgrade` command: this is used to bring all the local software up to date with those in the official repository.

Run `sudo apt-get update` followed by `sudo apt-get upgrade` with a new system to bring it up to date. It is a good idea to repeat this frequently to have the latest software on your computer.

Installing software. The default configuration of WSL does not have the software needed for creating documents with PreTeXt. There are a few commands to be run before you can get started.

- The program **git** is used to download the PreTeXt software onto your local computer. It is installed with the command `sudo apt-get install git`.
- The program **xsltproc** is used to create your readable documents. It is installed with the command `sudo apt-get install xsltproc`.

After these are installed, you are ready to set up PreTeXt.

Putting PreTeXt on your computer. Here are the steps necessary to get the PreTeXt software onto your computer:

- Make a new directory `mkdir mathbook`
- Make your own clone of the PreTeXt repository `git clone https://github.com/rbeezer/mathbook.git`
- Move to the new directory `cd mathbook`
- Initialize the new directory with `git pull`

This last command synchronizes your files with those in the official repository. You should run it frequently to keep your files up to date.

The simplest example. Here is a brief description of the use of WSL to create readable files. You, as the author, create the XML file. The system will contain an appropriate XSL file that translates your XML file to something readable.

Several editors come with WSL by default including NANO, PICO, VI, and VIM. Here are the steps to follow:

1. Type the command `cd` to align yourself in your home directory.
2. Use one of the editors to create a file called `hw.xml` (you could use the command `nano hw.xml`), and add the following text:

```
<?xml version="1.0" encoding="UTF-8" ?>
<mathbook>
    <article xml:id="hw">
    <p>Hello, World!</p>
    </article>
</mathbook>
```

3. Run the command `xsltproc mathbook/xsl/mathbook-html.xsl hw.xml` Upon completion, your should have a file called `hw.html`.
4. Now the tricky part: you want to view the `hw.html` file in a browser, but the usual Windows programs cannot see the files created within WSL. So we have to copy them to a place where they are visible. Fortunately, this is pretty easy to do. To put `hw.html` on your desktop, use the command `cp hw.html /mnt/c/Users/username/Desktop` (note that "username" must be replaced by your Windows user name). Once the file is on the desktop, a double click will open it in a browser.

The `edit-xsltproc-view` cycle just given may seem daunting at first blush. Some things that can help:

- Pressing the up arrow when at the command line displays the previously executed commands. Hitting the enter key while such a command is displayed executes it. This saves a lot of retyping.
- It is possible to define aliases to shorten commands. Your local Linux guru can show how this is done.
- It is possible to define scripts to shorten multiple commands. Your local Linux guru can show how this is done.

Appendix J

The PreTeXt Vagrant box

In this appendix, we describe how to install and use the official PreTeXt **Vagrant box**. Vagrant is a tool that allows for the automatic provisioning and version control of **virtual machines**. What this means for the PreTeXt user is that there is a quick, simple installation process that is nearly guaranteed to result in a correctly functioning, up-to-date PreTeXt installation. Moreover, this installation simulates the behavior of a native Linux installation while providing access to important files directly to the host operating system. Therefore much of this appendix is written with the assumption that most Vagrant box users are not already running a Unix-like host—i.e.they are running Windows. However, any PreTeXt user may benefit from a sanitized, centrally managed authoring environment.

Prerequisites for installation.

- 64-bit operating system (directions for determining this)
- VT-X enabled in your BIOS (Intel processors only)
- PuTTY or other SSH client (I recommend the portable install for PuTTY)

All versions of Windows 10 are 64-bit. If you are running Windows 7, open Windows Explorer and select **This PC** in the left-hand pane. Right-click and select "Properties". Examine the dialog for information about whether your Windows is 32- or 64-bit.

On Linux, execute `lscpu` and look for the line marked "CPU op-modes".

Mac OS X Lion (10.7) and up are 64-bit. If you are running Snow Leopard (10.6) or earlier, launch System Profiler and click Software (in the Contents section) to check for a 64-bit kernel.

VT-X may already be enabled on your machine, and it may not. If you have an AMD processor, there is nothing to do. Intel machines may need their BIOS reconfigured to run Vagrant. To enter the BIOS menu, restart your machine and watch closely. You will need to hold down a certain key during the boot sequence, often Escape, F2, or F12. As soon as you see the message, press the key and do not release until you see the BIOS menu. I usually have to restart more than once because I'm too slow.

Every motherboard has a different BIOS menu. Just look through all the menus until you find the VT-X option, then enable it and use the "Save changes and exit" option from the main menu. Be careful not to make other changes unless you know what you are doing.

Unix-like operating systems should have `ssh` installed. Windows users can download PuTTY here (I recommend the single-file installation under "Alternative Binary Files"—just download and run, no installer).

Installation.

1. Download and install Virtualbox
2. Download and install Vagrant

3. Open a console (Windows: press Windows key, type `cmd`)
4. Create a new directory `vagrant_demo` and change into it, using the command
   ```
   $ mkdir vagrant_demo
   $ cd vagrant_demo
   ```
5. Type the commands
   ```
   $ vagrant init daverosoff/pretext
   $ vagrant up
   ```
   This step takes between 30 and 60 minutes to complete. There is a lot of downloading. There will be some interesting messages, but once it gets going you don't have to watch.
6. Open PuTTY and connect to address 127.0.0.1, port 2222.
7. Answer "Yes" to the authentication message
8. Login user/pass: `vagrant`/`vagrant`
9. If you see the prompt `[vagrant@archlinux ~] $`, congratulations! You have successfully installed the PreTeXt Vagrant box.
10. Keep the PuTTY window open for use in the next section.

List J.0.1: Steps to install

A first example. Presumably, you already have a project underway. Use your usual file browser or command line to move or copy your project into the `vagrant_demo` directory you created earlier.

In your PuTTY window, invoke the commands

```
$ cd /vagrant

$ ls
```

and observe that your project is there, in your Vagrant box. The `/vagrant` directory in the Vagrant box is synced to the `vagrant_demo` directory you created earlier.

Do your normal compilation dance, including steps to build images, HTML, and PDF output. For me it might look like this:

```
$ cd m101
$ mkdir -p output/images
$ ~/mathbook/script/mbx -c latex-image -d output/images -f all src/index.mbx
$ xsltproc --xinclude --output output/ ~/mathbook/xsl/mathbook-html.xsl src/index.mbx
$ xsltproc --xinclude --output output/ ~/mathbook/xsl/mathbook-latex.xsl src/index.mbx
$ cd output
$ xelatex index.tex
```

After compilation is done, use your normal file browser to locate the output tree you created. Find some output and preview it, marveling at the simplicity of what has befallen your project.

Remark J.0.2 It is also possible to keep source in the `/home/vagrant` directory in the Vagrant box. Then the source will not be accessible from the Windows host. One possibility would be to use `git` to author in Windows and pull changes to the Vagrant box at compile time. I'm not sure why this would be better. Perhaps you don't want multiple copies of your source floating around your main disk drive.

You can halt or suspend your Vagrant box by issuing the `vagrant halt` or `vagrant suspend` commands, and destroy all vestiges of it with `vagrant destroy`. The latter does not remove the image from which the box is cloned, so starting another box later (perhaps in a different location) is much faster.

Known issues. The current build of the Vagrant box includes Sage, Asymptote, and WeBWorK via Docker as described at the wiki. Presumably, if your compilation dance requires a WeBWorK server, then you already have access to one and your normal compilation will still work. It does not yet include `jing` for validation; this will be included in the next release.

While I have tried to keep the virtual machine image as simple as possible, it is still fairly large, more than 5 GB, and any virtualization will demand a substantial amount of machine memory. Vagrant will run best if you have at least 8 GB of ram. I am interested to hear how well it works on less high-powered systems.

The `jing-trang` validation tool is not included in the present version.

Thanks for your interest in the PreTeXt Vagrant box. If you encounter errors, or if you think something is missing, please open an issue on Github.

Appendix K

GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE. The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS. This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover

Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING. You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY. If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using

public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS. You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties — for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS. You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS. You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS. A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION. Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION. You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE. The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING. "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents. To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with... Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index