

CONTEXTUS CLI

DRAFTING AND SUBMITTING CONTEXTUS DOCUMENTS

CONTENTS

Overview	1
Installation	2
Which version to use	3
Getting started	4
Anatomy of a command-line application	4
Add your API key	5
Set up a working directory.....	5
Preparing documents.....	7
Working with XML.....	7
Document Metadata	8
Simple Documents.....	9
Complex documents.....	11
Submitting documents.....	14
Validation.....	14
Submission.....	16

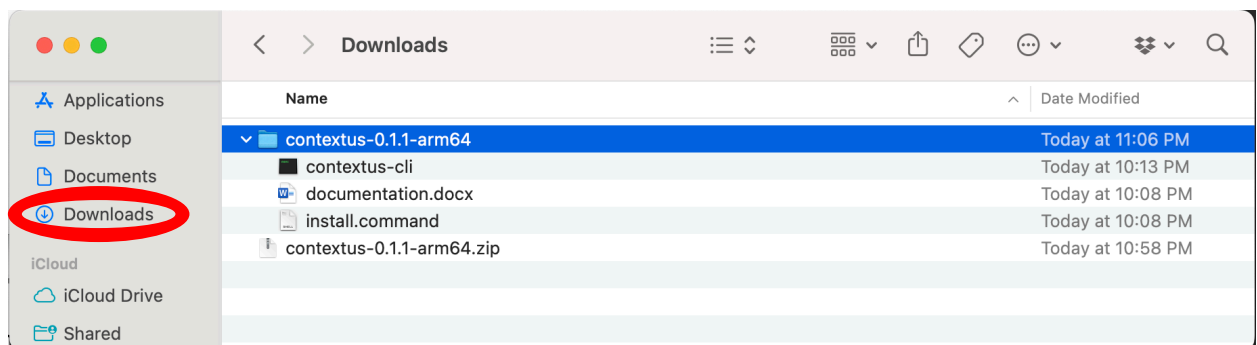
OVERVIEW

This document provides instructions for installing and running the Contextus command-line interface (CLI), an application that allows you to assemble Contextus documents locally (i.e., on your computer, as regular files), validate them, and submit them to contextus.org to become part of JMC's constitutional library. If you find that

these instructions omit anything, or are unclear in any way, please reach to hungerfordjustice@gmail.com with any questions that this document fails to answer.

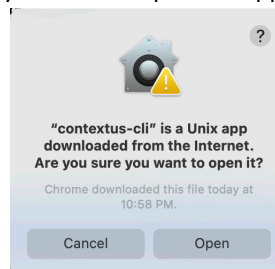
INSTALLATION

- Contextus-cli will be provided to you via email in the form of a zip archive, which will have the name contextus-0.1.2-arm64.zip or contextus-0.1.2-x64.zip depending on the type of processor your computer uses. The three numbers (the version) may periodically change as bugs are fixed or new features are added.
- Download the zip file to any folder, open it in your finder (typically it will end up in `Downloads` if you download it from a web browser) and double click on it.



This will create a folder in the same directory with the same name (but no `.zip` extension). Inside this directory you will find three files: `contextus-cli`, `documentation.docx` (this file), and `install.command`.

- Right click (or hold the control button while clicking) `contextus-cli` and select "Open." A window should open asking you to confirm that you want to open the application. Confirm by clicking "Open":

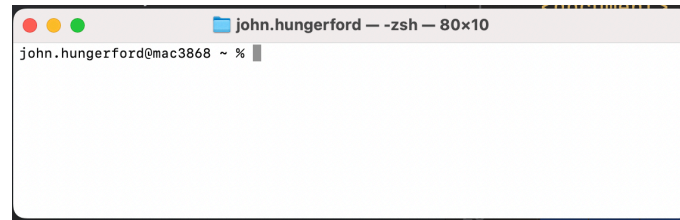


A terminal window should open which will display a bunch of text. When it stops (you should see something like `[Process completed]` at the bottom), close it. If you see the message like *"contextus-cli" is damaged and can't be opened. You should move it to the Trash*, right click the `contextus` folder in the Finder window and select Services -> New Terminal at Folder. When the terminal opens, type the following command at the `%` prompt and hit enter

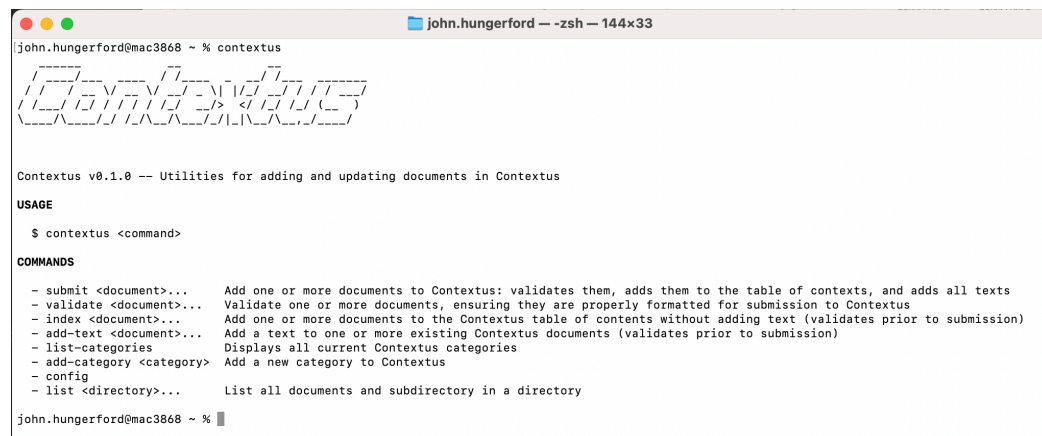
```
xattr -dr com.apple.quarantine contextus-cli
```

Now try to run it again.

- Repeat the same process with `install.command`. Again, if you see a message that it is damaged, open a terminal window and run: `xattr -dr com.apple.quarantine install.command`, and try opening it again. Once it runs, contextus should be installed.
- To confirm that installation worked, open a new terminal window but pressing command + space (press and hold the “command” button next to the space bar, and then press the space bar while still holding the command button). In the pop-up window, type “Terminal” and press enter. This should bring up a new window that looks like this:



- At the prompt (the “%” sign), type `contextus`. The terminal window should display the following:



If you find the text too small to read comfortably, hold the command button and press +/- a few times until it is a desired size. Conversely hold command and press -/_ to make the text smaller.

WHICH VERSION TO USE

As mentioned above, contextus-cli is provided both for older macs, which use Intel chips, and new ones, which use a more advanced M1/M2 chips that have a sufficiently different architecture that executables compiled for the one typically cannot be run on the other. To figure out which version you'll need, click the apple icon on the top left-hand corner of your screen and select "About This Mac." A window should pop up that looks like this:



Look for the line that begins with “Chip:” – it will display either some version of “Apple M1”/“Apple M2” or some longer description that includes “Intel” somewhere. If your chip is the former (Apple M_), your computer is said to have “ARM” architecture, if the latter, it has “x64” architecture.

You should be provided two versions of the contextus-cli package, one that ends with ...-arm64.zip and one that ends with ...-x64.zip. Use the one that corresponds with the architecture described above. Or for simplicity:

- If your chip is Apple M1 or Apple M2, use contextus-X.X.X-arm64.zip
- If your chip is Intel, use contextus-X.X.X-x64.zip

GETTING STARTED

ANATOMY OF A COMMAND-LINE APPLICATION

You will be using contextus commands to upload documents to the contextus website. Contextus commands have a consistent structure:

```
% contextus [COMMAND(s)] [OPTIONS] [ARGUMENTS]
```

[COMMAND(s)] is where you will type the actual command. Supported commands are `index`, `add-text`, `validate`, `submit`, `list-categories`, `add-category`, `config`, and `list`. In some cases, there will be sub-commands. For instance, you will use the command `contextus config api-key` to view or update your API key (see below)

[OPTIONS] are statements you may or may not include when invoking a command that provide some additional configuration. Options will always begin with either one or two dashes: `--some-option`, or `-s` for short. Options will usually require some additional information to be provided right after. For instance, to set your API key, you will use the command `contextus config api-key --set ABCDEFG`, where “ABCDEFG” is the value you want to set the key to.

[ARGUMENTS] are the main inputs to the command you are invoking. To submit a document, for instance, you provide the document file as an input:

```
% contextus submit my-document.xml
```

If you try to run a command with an incorrect argument or without some required option or argument, you should see an error message explaining what was wrong:

```
john.hungerford@mac3868 ~ % contextus submit non-existent-file
No file or directory at path non-existent-file
```

You can run any command with the `--help` option to see documentation about what options and arguments are expected. For instance,

```
john.hungerford@mac3868 ~ % contextus submit --help

  /-----\  /-----\  /-----\  /-----\  /-----\
 /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /
/  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /  /
\  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \
 \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \  \

Contextus v0.1.0 -- Utilities for adding and updating documents in Contextus

USAGE

  $ contextus submit <document>...

DESCRIPTION

  Add one or more documents to Contextus: validates them, adds them to the table of contexts, and adds all texts

ARGUMENTS

  <document>...
    A user-defined piece of text.

  Path to an XML document to be processed, or a directory with XML files to be processed. Uses current working
  directory by default.

  This argument may be repeated zero or more times.
```

ADD YOUR API KEY

To allow contextus-cli to access contextus.org, you will need to provide it with an “API key,” which is a kind of password that authorizes it to make changes to the Contextus library. You should have received a key from your contextus-cli maintainer. If you have not, reach out to hungerfordjustice@gmail.com.

The key should look something like this: 35ec15f3a129e30a

Once you have your key, run the following command, replacing the key value with the one you have received:

```
% contextus config api-key --set 35ec15f3a129e30a
```

Confirm that the key has been set by running the command without the `--set` option:

```
% contextus config api-key
```

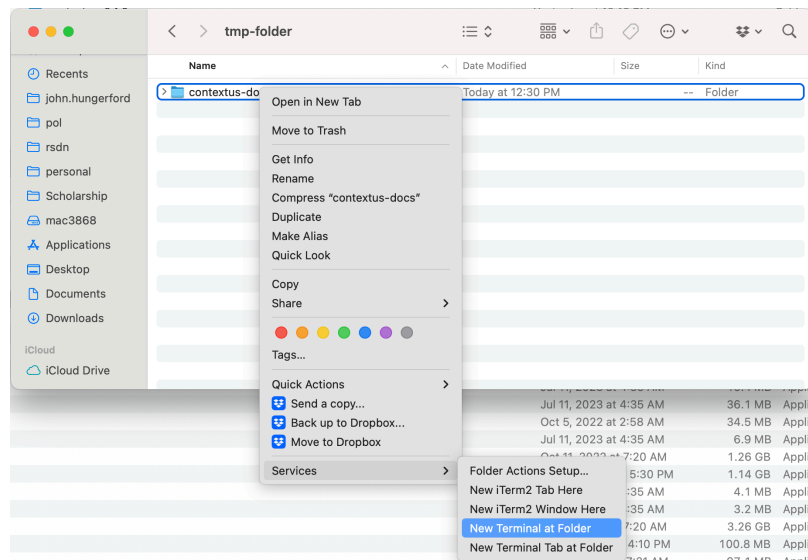
The terminal should display the value you just provided. If it does not, please contact your contextus-cli maintainer.

SET UP A WORKING DIRECTORY

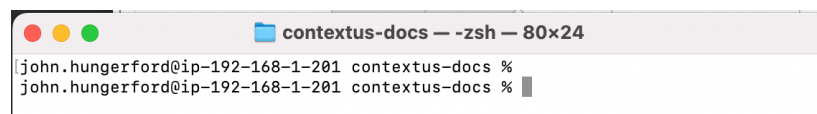
Your document submission workflow will consist mainly of editing documents on your own computer, which you can then submit either individually or in batch to contextus.org. It will be easiest for you if you maintain a single folder where you keep all these documents.

Create a new empty folder in some convenient location (e.g., a folder called contextus-docs inside of your Documents folder). This will be your “working directory” which where you will keep all your contextus documents and where you will run all contextus-cli commands from.

Now let’s open a terminal for this folder. Right click on the folder in the finder window, and click on **Services -> New Terminal at Folder**



This should open a new terminal window. You will notice, however, that to the left of the command prompt (%) the name of the folder is displayed. Any command you run from this terminal will execute relative to this folder.



To see how this works, lets run our first real command. Type the following command in your new terminal window and hit enter:

```
% contextus new-document "Example Title (1860)"
```

If everything is working, it should respond with “Created new document: example-title-1860.xml”

Contextus has just created a template document for you within your working directory. To confirm that it was generated, look in your working directory in the finder. You should see example-title-1860.xml in the folder.

You can also use contextus-cli to examine your working directory. Run the following

```
% contextus list
```

It should respond with a list of documents in the folder, which at this point will simply be the example document.

PREPARING DOCUMENTS

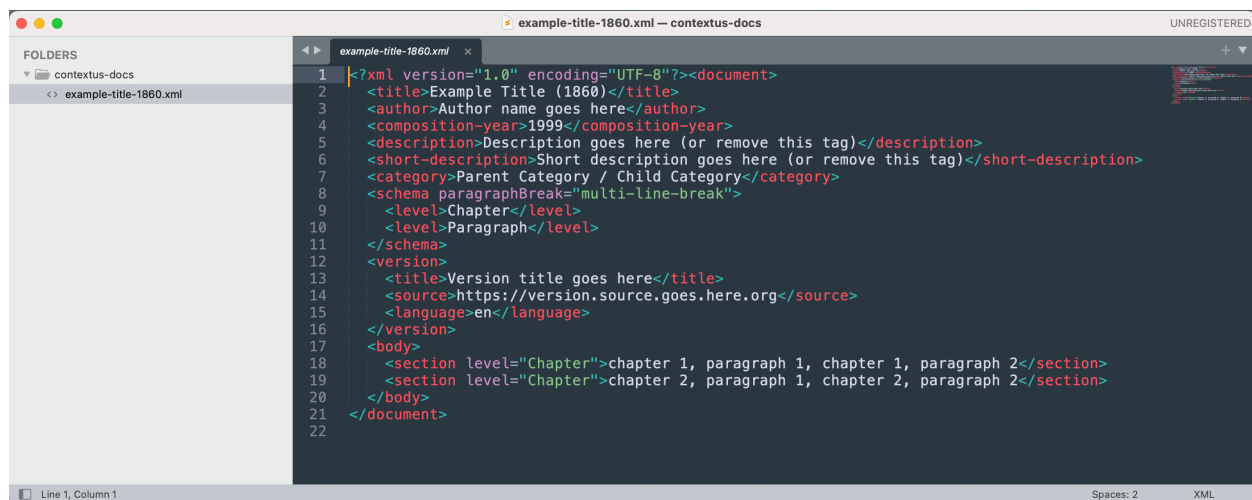
WORKING WITH XML

To prepare a contextus document, you will be using a text format called “XML.” XML is a more general form of HTML, which is the text format used by web pages. To work with XML easily, it helps to have a text editor that has XML support. I recommend using Sublime text editor, which is free and has decent XML features that don’t get in the way of usability. You can download Sublime here:

https://www.sublimetext.com/download_thanks?target=mac

When Sublime has installed, open it and select **File -> Open**. Now, instead of opening a file, navigate to the working directory you created in the last section, select it (don’t enter it!) and click the **Open** button. This should open a panel on the left-hand side of the window titled “Folders,” which should display your working directory and its contents. Now you can open any file saved in this directory without going back to your finder.

Open the example document we created in the previous section, which you should see in the left-hand panel. You should now see the following in the editor panel:



```
1 <?xml version="1.0" encoding="UTF-8"?><document>
2 <title>Example Title (1860)</title>
3 <author>Author name goes here</author>
4 <composition-year>1999</composition-year>
5 <description>Description goes here (or remove this tag)</description>
6 <short-description>Short description goes here (or remove this tag)</short-description>
7 <category>Parent Category / Child Category</category>
8 <schema paragraphBreak="multi-line-break">
9   <level>Chapter</level>
10  <level>Paragraph</level>
11 </schema>
12 <version>
13   <title>Version title goes here</title>
14   <source>https://version.source.goes.here.org</source>
15   <language>en</language>
16 </version>
17 <body>
18   <section level="Chapter">chapter 1, paragraph 1, chapter 1, paragraph 2</section>
19   <section level="Chapter">chapter 2, paragraph 1, chapter 2, paragraph 2</section>
20 </body>
21 </document>
22
```

As you can see, much of the document is filled with chunks of text in the form of either `<some-word>` or `</some-word>`. These are called “tags.” The tag that has no leading slash (e.g., `<some-word>`) is called an “opening tag,” and the kind with a leading slash is called a “closing tag.” Every opening tag must eventually be followed by a closing tag with the same label. These opening closing tags enclose more text (this can be plain text, more tags, or both), which these tags are intended to describe. For instance, you will see that the `<title>` and `</title>` tag pair enclose the title of the document, and the `<author>` and `</author>` enclose the author name (for now a placeholder).

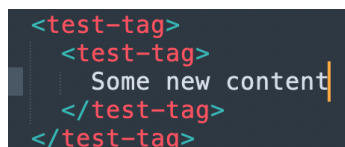
You can also see some cases where within the opening tag are words followed by an equal sign and some text enclosed by quotation marks. For instance, the opening schema tag has `<schema paragraphBreak="multi-line-break">`. The word `paragraphBreak` is called a “tag attribute” (it is an “attribute” of the schema tag), and the quoted text `“multi-line-break”` is the “attribute value.” This is a way to provide additional information within the tag that can be used by `contextus-cli` when parsing and submitting documents.

Every part of the contextus document must be properly tagged prior to submission. All the metadata (i.e., the information *about* the document, such as the title, author, publication date, etc.) must be placed in appropriate tags, and all the actual text must also be enclosed in the tags that describe where that text should be in the documents *structure* (e.g., is it a book, chapter, verse, or paragraph?).

Sublime makes this process a little easier. You can already see that it provides “syntax-highlighting” to differentiate the tags from the rest of the text. To see how, start by adding a new line between the opening `<document>` tag and the opening `<title>` tag. In the new line, start to add a new tag, but don’t include the closing “`>`”: “`<test-tag`”. You will see it is automatically colored red like the other tags, you will also see some new red highlighting before the `<title>` tag on the next line. This red highlighting is indicating that there is invalid syntax: you cannot start a new tag before you’ve completing the previous. Sublime will thus help you spot accidental mistakes.

Next, finish the opening `<test-tag>` tag and then type “`</`”. Completing the opening tag should remove the error highlight. After typing “`</`” you should find that rest of `</test-tag>` is immediately filled in for you. Sublime’s awareness of XML syntax allows it to anticipate which tag must be closed next and autocomplete closing tags for you.

Finally place your cursor between the “`>`” and the “`<`” and hit enter. Sublime should split the line into three separate lines, with `<test-tag>` and `</test-tag>` on the first and third line, and the cursor on the second. Moreover, while the `<test-tag>` and `</test-tag>` will share the same indentation, the cursor will be indented one tab space relative to them. Sublime will thus automatically indent tagged content to help you see the structure of the XML document more easily. If you repeat this process where the cursor ended up, the next tag content should be indented even further:



```
<test-tag>
<test-tag>
  Some new content
</test-tag>
</test-tag>
```

DOCUMENT METADATA

Contextus XML documents have the following structure

1. `<document>` tag: the entire document must be enclosed in `<document>` and `</document>`. This should be the outermost tag, except for a special `<?xml ...?>` tag that is not required but is generated by the “new-document” command (it does not provide any necessary or even useful information)
2. Metadata tags: all of the tags prior to the `<body>` tag provide information about the document
3. `<body>` tag: this is where the actual text goes, along some structural tags

The metadata tags are mostly self-explanatory, but here are some useful notes about some of them:

- `<title>` and `<author>` are required
- `<composition-year>`, `<description>` and `<short-description>` are optional. `<composition-year>` must be a number
- `<category>` must be in the form Parent Category / Child Category, where the Child Category is a category within the Parent Category. You must provide a valid `<category>` to submit a document

- `<schema>` is required
 - The `<level>` tag within `<schema>` indicates a structural level of the document. These should be ordered from outer to inner, so


```
<level>Book</level>
<level>Chapter</level>
<level>Paragraph</level>
```

 and not the other way around. These levels will need to correspond to the `<section>`s within the `<body>` tag.
 - The `paragraphBreak` attribute in schema determines how text will be parsed by `contextus-cli`. More specifically, it determines how it will parse the *last level* from the text. Typically, the last level of a text will be either “Paragraph” or “Verse.” You will not want to have to tag each of these individually, so instead `contextus-cli` gives you the option of either interpreting line breaks or, more typically “multi-line breaks” (i.e., one or more blank lines between chunks of text) as indicating divisions between sections.
- `<version>` is required and contains the metadata about the version of the text used for the document. Within `<version>` you will need:
 - `<title>`: the title of the version, e.g., Project Gutenberg
 - `<source>`: the URL of the web page the text was taken from
 - `<language>` is optional and should just be `<language>en</language>` if included

SIMPLE DOCUMENTS

There are two kinds of document structures: simple and complex. A “simple” document has a perfectly consistent structure: it can have chapters and paragraphs; or books, chapters, sections, sub-sections, and paragraphs; or verses and lines, but it cannot have one book with chapters, sections, subsections, paragraphs, and then another book with just chapters and paragraphs. Moreover, its sections cannot have individual titles. It can only have “Chapter 1,” “Chapter 2,” “Chapter 3,” etc., and not “Chapter 1: the boy who lived,” “Chapter 2: the vanishing glass,” etc.

The structure of a simple document is determined entirely by the `<schema>` section. Every section within the `<body>` tag of a simple document must conform to the `<level>`s in `<schema>`. The simplest of all simple documents will have only one level, say, “Paragraph.” Such a document will not need any tags within the `<body>` tag, but only plain text. Paragraphs will be distinguished by multi-line breaks within that text. E.g.,

```
<document>
...metadata tags...
<body>
  This is paragraph one.

  This is paragraph two.
</body>
</document>
```

Any additional levels must be enclosed by a `<section>` tag, however. Moreover, these section tags must include a “level” attribute indicating which level it corresponds to. This is to make sure you know what level you are adding

text to (contextus-cli will not let you submit a document with incorrectly labelled levels). To add chapters to the above example you would do the following

```
<document>
...metadata tags...
<body>
  <section level="Chapter">
    This is chapter 1, paragraph one.

    This is chapter 1, paragraph two.
  </section>
  <section level="Chapter">
    This is chapter 2, paragraph one.

    This is chapter 2, paragraph two.
  </section>
</body>
</document>
```

Finally, we can add books:

```
<document>
...metadata tags...
<body>
  <section level="Book">
    <section level="Chapter">
      This is book 1, chapter 1, paragraph one.

      This is book 1, chapter 1, paragraph two.
    </section>
    <section level="Chapter">
      This is book 1, chapter 2, paragraph one.

      This is book 1, chapter 2, paragraph two.
    </section>
  </section>
  <section level="Book">
    <section level="Chapter">
      This is book 2, chapter 1, paragraph one.

      This is book 2, chapter 1, paragraph two.
    </section>
    <section level="Chapter">
      This is book 2, chapter 2, paragraph one.

      This is book 2, chapter 2, paragraph two.
    </section>
  </section>
</body>
</document>
```

```
</body>
</document>
```

COMPLEX DOCUMENTS

For complex documents, you want to think about the document as being broken down into “sub-documents.” Each sub-document will have its own title and, optionally, its own schema. Each sub-document can either have more sub-documents or can be a simple document. Sub-document schemas are optional because if they are omitted, the schema from the parent document will be used. If no parent document has a schema, the schema from the main metadata section will be used.

Let’s consider the simplest form of a complex document: a book with a preface and/or named chapters. The document would look something like the following:

```
<document>
  ...metadata tags...
  <schema>
    <level>Paragraph</level>
  </schema>
  <body>
    <section>
      <title>Preface</title>
      This is preface, paragraph one.

      This is preface, paragraph two.
    </section>
    <section>
      <title>Chapter 1, the Boy who Lived</title>
      This is chapter 1, paragraph one.

      This is chapter 1, paragraph two.
    </section>
    <section>
      <title>Chapter 2, the Vanishing Glass</title>
      This is chapter 2, paragraph one.

      This is chapter 2, paragraph two.
    </section>
  </body>
</document>
```

A few things to note in this document:

- Each sub-document is defined by a <section> tag, similar to a simple document but notably with the “level” attribute.
- Sub-document <section>s include <title> tags

- There is only one schema applied to the entire document that specifies a single level: paragraph. There is no need to specify any other levels, since the rest of the document structure is captured by the <title>s given to the various sub-document sections

Let's make this document a little more complicated. Suppose the book is separated into parts. We will need to add sub-documents for each part and place the chapter sub-documents inside of them. So long as each chapter is still divided only into paragraphs, we can use the same single schema.

```
<document>
...metadata tags...
<schema>
  <level>Paragraph</level>
</schema>
<body>
  <section>
    <title>Preface</title>
    This is preface, paragraph one.

    This is preface, paragraph two.
  </section>
  <section>
    <title>Part 1</title>
    <section>
      <title>Chapter 1, the Boy who Lived</title>
      This is chapter 1, paragraph one.

      This is chapter 1, paragraph two.
    </section>
    <section>
      <title>Chapter 2, the Vanishing Glass</title>
      This is chapter 2, paragraph one.

      This is chapter 2, paragraph two.
    </section>
  </section>
  <section>
    <title>Part 2</title>
    ... (chapters for Part 2 go here)
  </section>
</body>
</document>
```

Now let's suppose that each chapter were divided into sections called "Divisions." These divisions are unnamed, so we can use the regular schema to describe them, and contextus will automatically number them for us. Let's suppose, however, that the preface does not have "Divisions" but only paragraphs. In this case, we'll define

“Divisions” and “Paragraphs” in the schema for the whole document, which will be inherited by *all* sub-documents, but when we define the preface, we’ll provide a separate schema to override it:

```
<document>
  ...metadata tags...
  <schema>
    <level>Division</level>
    <level>Paragraph</level>
  </schema>
  <body>
    <section>
      <title>Preface</title>
      <schema>
        <level>Paragraph</level>
      </schema>
      This is preface, paragraph one.

      This is preface, paragraph two.
    </section>
    <section>
      <title>Part 1</title>
      <section>
        <title>Chapter 1, the Boy who Lived</title>
        <section level="Division">
          This is chapter 1, division 1, paragraph one.

          This is chapter 1, division 1, paragraph two.
        </section>
        <section level="Division">
          This is chapter 1, division 2, paragraph one.

          This is chapter 1, division 2, paragraph two.
        </section>
      </section>
      <section>
        <title>Chapter 2, the Vanishing Glass</title>
        <section level="Division">
          This is chapter 2, division 1, paragraph one.

          This is chapter 2, division 1, paragraph two.
        </section>
        <section level="Division">
          This is chapter 2, division 2, paragraph one.

          This is chapter 2, division 2, paragraph two.
        </section>
      </section>
    </section>
  </body>
</document>
```

```

    <section>
      <title>Part 2</title>
      ... (chapters for Part 2 go here)
    </section>
  </body>
</document>

```

Note that now that the chapter schema has an additional level, the “Division” section has to be provided explicitly with a `level=“Division”` attribute.

FORMATTING TEXT

In addition to metadata tags and structural tags, contextus-cli supports several different tags for formatting text. These tags, which can be used anywhere within the actual text of your document, are as follows:

- `bold text` — **bold text**
- `emphasized, or italicized text` — *emphasized, or italicized text*
- `<u>underlined text</u>` — underlined text
- `struck through text` — ~~struck through text~~
- `^{superscripted text}` — ^{superscripted text}
- `_{subscripted text}` — _{subscripted text}
- `<note number=“1”>`this text will appear when you click on a superscripted “1” and can be used as a way of including footnotes`</note>` — can’t be rendered in a word document

Note that if you miswrite these, validation will not necessarily identify your mistakes. How these mistakes will affect the appearance of document text on the website is not easy to predict.

SUBMITTING DOCUMENTS

VALIDATION

Now that we’ve seen how to create documents, let’s go over how to use contextus-cli to upload them to contextus.org. Before trying to upload a document, however, it’s helpful to simply check that you’ve drafted it properly. There are a lot of different tags that you need to get right for everything to work, and the document structure and schema must match up properly. To validate that a document is create correctly, you can run the following:

```
% contextus validate [document-file-name]
```

Where “document-file-name” should be the full file name (not the title!) of the document you’ve created. Let’s try it with the example document we generated before:

```
% contextus validate example-title-1860.xml
```

When I try to run this I get the following error:

```
Unable to process example-title-1860.xml
Invalid Contextus document:
  Unable to find category Parent Category in index
```

This is because the place-holder category we used was Parent Category / Child Category. Contextus-cli looked for Parent Category in contextus and found that it wasn't there. We need to change it to a category that does exist. To do this, let's look up the categories that are in the index using the following contextus-cli command:

```
% contextus list-categories
```

This should return a list of categories indicating their hierarchy through their level of indentation. Here's a small subset of the current category list:

```
American Priors
  Religious Tradition
    Old Testament
    New Testament
  Ancient Political Theory
    See more
  Modern Political Theory
  Legal Precedents
  Other
Toward Independence
  Colonial America
    Charters and Legal Texts
    Sermons and Speeches
    Pamphlets and Treatises
```

Based on the indentation, we see that some valid <category> values would be (among others):

- American Priors / Religious Tradition / Old Testament
- American Priors / Modern Political Theory
- Toward Independence / Colonial America / Sermons and Speeches

Let's change our category to Toward Independence / Colonial America / Sermons and Speeches and see if that works. When we try running our validate command again (see above), we should now get the following output:

```
Valid Contextus document: Example Title (1860) (example-title-1860.xml)
```

Note that you don't have to specify the document explicitly, but can just run:

```
% contextus validate .
```

The period "." in the above command stands in for "the current directory," which in this case is your working directory. When you provide a directory as an argument to contextus validate, contextus-cli validates every xml file it finds in that directory. The above command should therefore validate every document in your working directory.

SUBMISSION

Once you have a valid contextus document with all the required content, it is easy to submit it to the website. Simply run:

```
% contextus submit [filename]
```

This will do three things: validate the document, index it (add it to the table of contents), and add the text. Indexing and adding text are considered two different operations, and can be done separately using the commands `contextus index [filename]` and `contextus add-text [filename]`. Running the submit command is therefore more or less equivalent to running:

```
% contextus validate [filename]
% contextus index [filename]
% contextus add-text [filename]
```

Like the `validate` command, you can provide a directory for the three other commands `index`, `add-text`, and `submit`.

All three commands also automatically run validation, so you will not be able to accidentally index or add text for an invalid document.