A Guide to Diderot

September 2, 2019

Chapter 1

Course Staff's Guide to Diderot

This chapter describes common course management workflows for instructors and teaching assistants.

1.1 Course Creation

To create a course on Diderot please email us at [aada,umut]@cs.cmu.edu. As part of the course creation process, we will determine if your course may be private (to your class only) or public (accessible to anybody with a Diderot account). For a private course, we will set a passkey that can be used for students to register for the course.

1.2 User Accounts

In principle anybody may sign-up for Diderot. They will need a valid email address from an approved domain, e.g., $*$.cmu.edu. The set of approved domains is determined by allowed domains of existing courses.

When a student logs onto Diderot, they will have access to all public courses and could register for a private course with the correct passkey.

1.3 Course Home Page

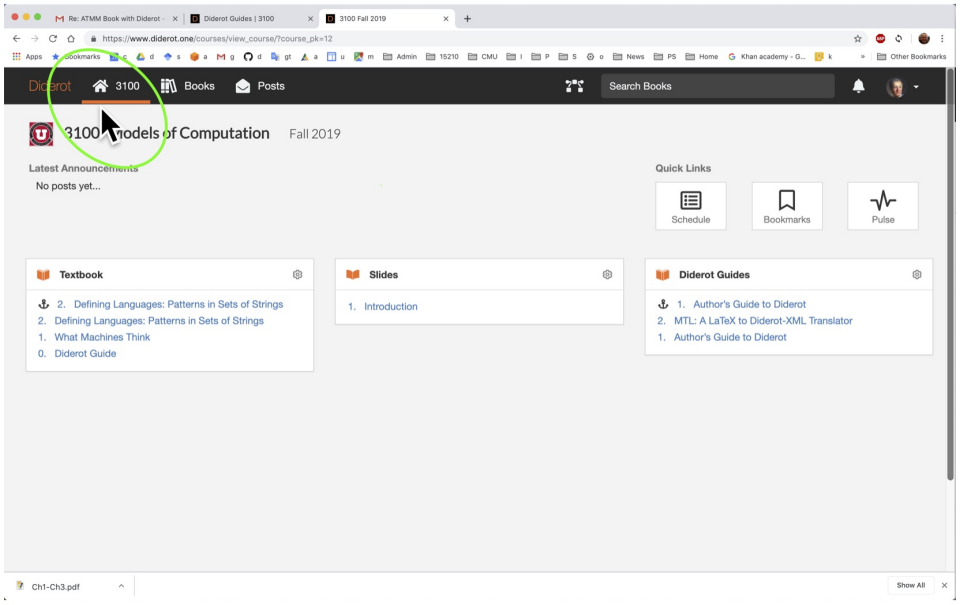
The Zones. Course home page, shown below, consists of four zones.

1. A toolbar, at the top
2. A "news" or "Latest Announcements" area on the top left quadrant.

3. Quick links on the right top right quadrant.

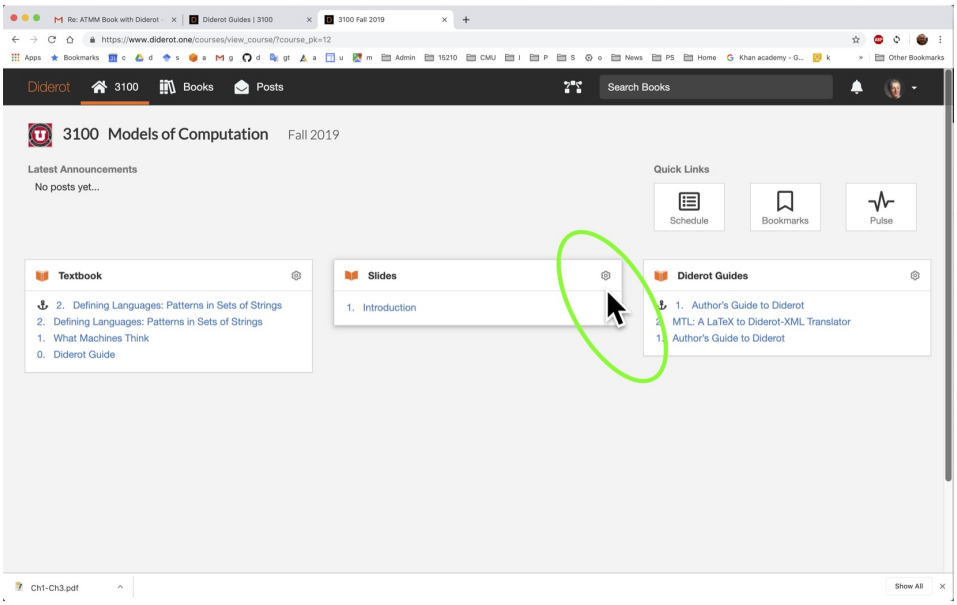
4. The books, an the lower half.

You can go to the course home page by pressing the "Home" button at the top right corner.



1.4 Book Management

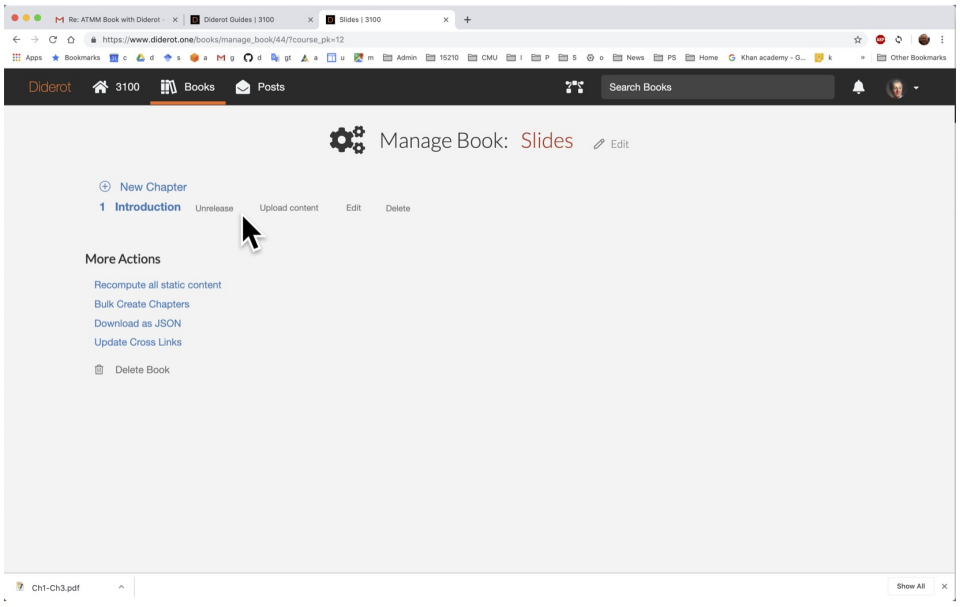
Finding Book Management. As an author, you will spend much of your time in the "Book Management" page. A quick way to get to this page is to go to the course home page and to press the settings ("wheel") icon for the book that you want to manage.



The book management page offers functionality for

- creating chapters,
- uploading content onto chapters,
- realeasing/unreleasing chapters,
- deleting chapters,
- additional more advanced operations, which we will skip for now.

To see the actions that you can perform on a chapter options, move the mouse over to that chapter.



Exercise 1.1. Create a book called "Slides" for uploading some slide decks.

1. Go to "Manage Books" from the "Links" on the center right, to the left of search bar on top.
2. Create the book "Slides".

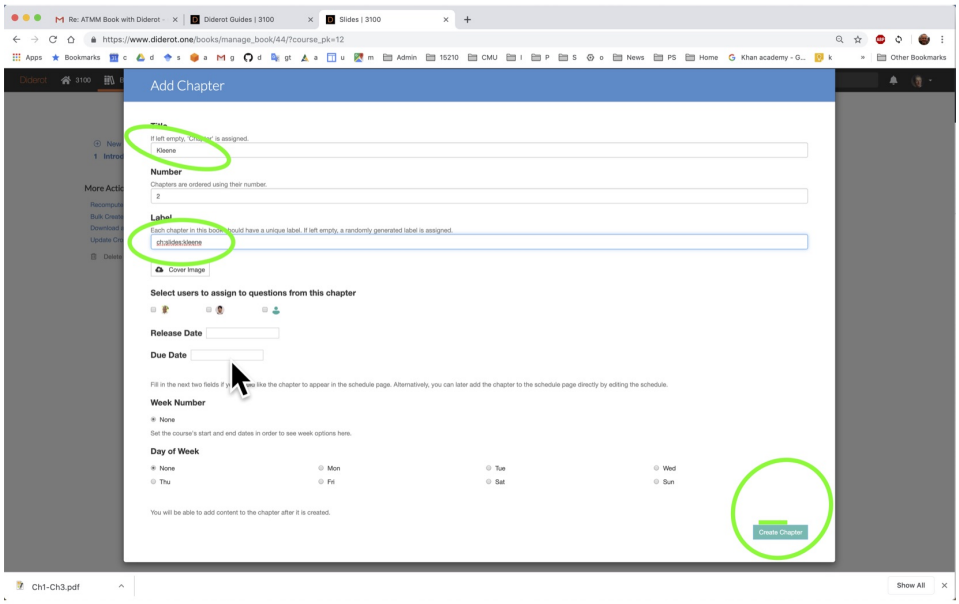
1.5 Chapters

Chapter Creation. You can add a chapter to a book.

- Go to course home page.
- Find the book that you want to edit. You can reach the book management screen from the "Manage Books" link from the "Links" button on the center right, to the left of search bar on top.
- Press "Create chapter" and follow the instructions. To create a chapter in its most basic form, you can leave all of the rest of the fields empty. The additional field are discussed below.

When creating a chapter, assign the chapter a unique number and a unique label, e.g., `ch:kleene`. The label needs to be unique only within the book—no two chapters of the

same book can have the same label. If the chosen label does not meet this criteria, you will receive an error message and can choose another one.



Exercise 1.2. Create a chapter for a slide deck and give it the label `ch:test-slides`.

Released versus Unreleased Chapters. A chapter is either released or unreleased. ***Released*** chapters are visible to the students. ***Unreleased*** chapters are not visible to the students but are visible to course staff.

This is designed to allow for a "feedback cycle" before releasing chapters to students. For example, I typically upload my lectures a bit ahead of time and before releasing them, ask my TAs to look over them and give feedback. To give feedback, the TA's simply select the relevant atom and create a feedback, e.g., they might note a typo. I then fix these problems, reupload the chapter, and then release it.

Release Dates and Schedule. You can assign release dates to your chapters and this will automatically construct (and maintain) a schedule for you. Let's skip this step for now. So leave these fields empty. You can edit chapters to add schedule information later.

User Assignments. You can "assign" users to a chapter. The intention here is to designate ***discussion chairs***, who are usually TAs for each chapter. They will be assigned the questions asked on that content.

I don't know who I will assign the chapters to at the time of creation and therefore leave this blank. I or the TAs later edit the chapter to assign the discussion chairs.

Upload Window. To upload contents into a chapter, move the mouse over the chapter and select upload content. Here you can upload either an XML file or a PDF.

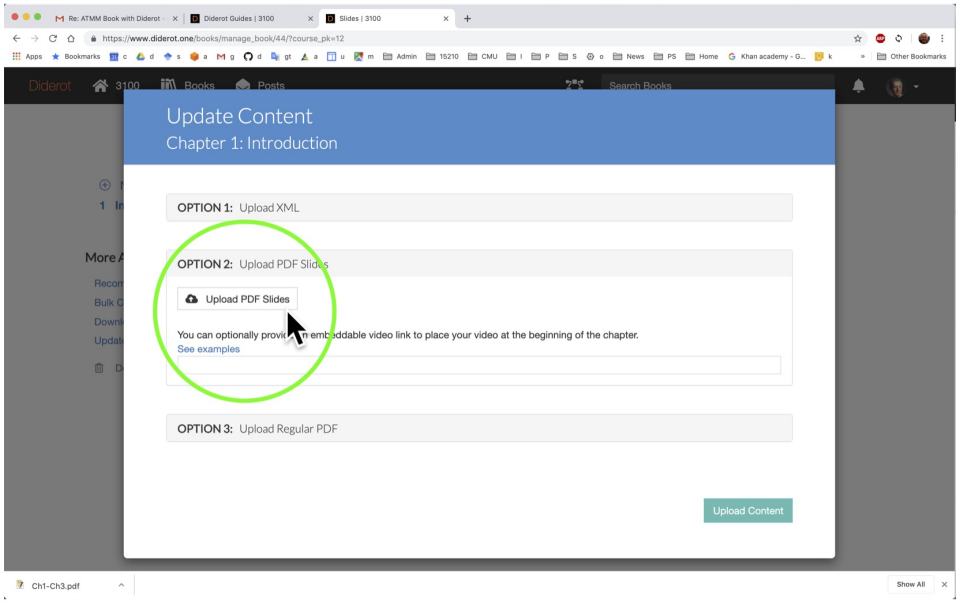
PDF Uploads. To upload a PDF document bring up the upload window and select

- "Upload PDF Slides", or
- "Upload Regular PDF."

Then select the PDF file to upload, and press "Upload Content".

Now you will see a spinning wheel. This will take some time maybe a minute or two. What is going on is that the PDF is split into pages and the text is extracted from it. When it is finished, click on the chapter and you should be able to view it.

As an example, upload your second slide deck from your Fall 2018 site to the chapter that you have created above.

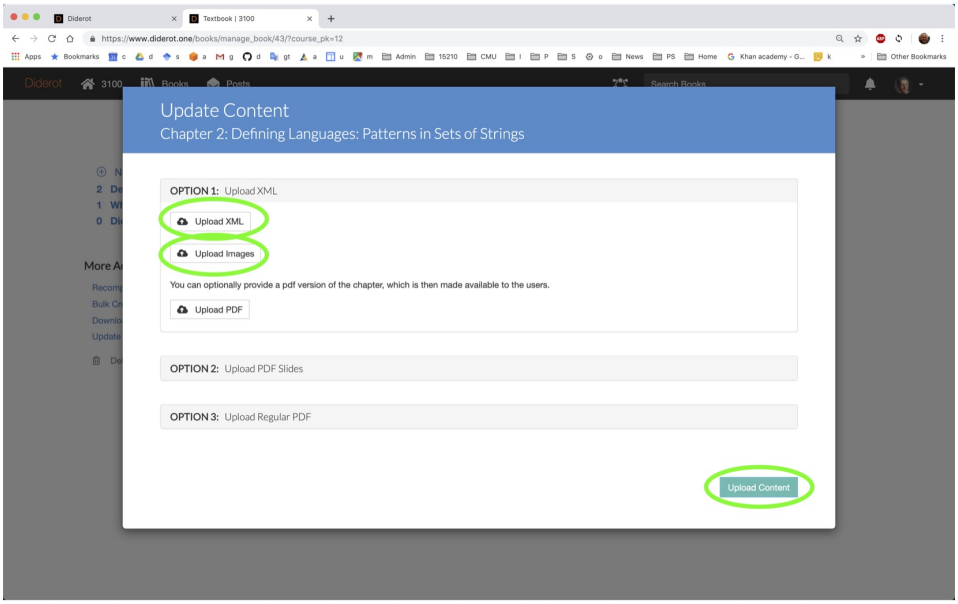


XML Uploads. To upload a Diderot XML, first use the DC compiler to generate one. You can generate Diderot XML from LaTeX or from Markdown. Next, press the "Upload XML" and select the XML file that you want to upload.

If your chapter includes images, you can upload the images along with your XML by pressing "Upload Images" and then selecting all of the images (multiple images can be selected) that you want to upload.

Now press "Upload Content". You will see a spinning wheel. This will take some time maybe a minute or two. What is going on is that your XML document is parsed into a tree of a chapter, sections, and atoms and everything is linked so that the images and document links now point to their right places.

When the spin stops, click on the chapter and you should be able to view it.



Important (Unreleased chapters are not searchable). When a PDF/XML document is uploaded, its contents is stored into an index, allowing it to be searched via the search bar when the chapter is released. Unreleased chapters will not be searchable (so students won't see unreleased content).

Note. Currently, pdf documents themselves are shown as JPEG images. We are in the process of changing this so that they are shows an PDFs. This should be ready soon.

Interactive Chapters. Once a chapter is uploaded, it becomes interactive.

- If unreleased, course staff may send feedback and open other discussions.

- If released: all the students and staff may send feedback, ask questions, take private notes on the content, etc.

Updating a Chapter. For unreleased chapters, simply re-upload the contents. This will recreate the chapter and delete all prior discussions on it.

For released chapters, you can unrelease and re-upload, but this will delete all content, including student created content (private notes, questions, etc). Unreleasing and re-uploading could be thought as the same as deleting the chapter and recreating it. Because it deletes all related content, we discourage using this feature (after a course starts). If you don't currently have students and are developing content, you may release chapters, and re-upload them as you wish.

There is only one reliable way to update released content without deletion or without unreleasing it: assign to the chapter, all sections, and all groups, and atoms a label. This allows Diderot to match the content being uploaded to existing content without ambiguities. Otherwise, there can be possible ambiguities, because, for example, Diderot doesn't know which updated section corresponds to which existing section.

1.6 Books

Create a book. You can create new books and as many as you want. I usually create one for textbook, one for recitations, one for assignments, one for additional materials (misc). Many courses create a "Slides" book.

To create a book go to "links" to the right of the search bar and then press manage books. You will give your book a unique label (e.g, "textbook", "recitations", "misc"...) and a unique"rank" which is the order it will be shown but is otherwise immaterial.

By default, a book is selected to be a ***booklet***, which consists of a sequence of chapter. If you want the book to contain "parts" as an organization element above chapters, uncheck the "booklet" box.

A book can contain PDFs or XMLs (generated by a compiler).

Example 1.1. In the algorithms (15210) course, the main textbook has parts but the rest of the books don't. So all the books except for the textbook are booklets.

Editing a Book. To edit the book, create and upload chapters, go to "Manage Books" and select your book. Here you can edit various properties of the book, create chapters upload PDF content , and upload XML content .

1.7 Course Registration

Students can self-register for a Diderot course by using a passkey created by the instructor (and provided to the students). To register, the students will need to use an email address from an instructor-speficied ***domain***. For example, at Carnegie Mellon, an instructor would typically allow a student to register by using their andrew.cmu.edu email only. Instructors may invite teaching assistants and other instructors to their course.

To invite users, you can upload an invitation list in CSV format.

1. Go to "Manage Users" under "Links"
2. Click "Add Users from CSV".

The CSV files must have the following columns

- Last Name
- Preferred/First Name
- Email
- Role, which is either 'Instructor', 'TA', or 'Student'.

The CSV file may contain additional columns.

Example 1.2 (User CSV)**.** The following CSV file could be used to invite an instructor and additional TA's to a course.

```
Last Name, Preferred/First Name, Email, Role
Srinathan, Ramgopal, ramu@cs.diderot.edu, Instructor
Tesla, Nikola, tesla@diderot.edu, TA
Hendrix, Jimmy, jimmy@gmail.com, TA
Santana, Carlos, jrjacobsoniii@gmail.com, TA
```

Chapter 2

DC: Diderot Compiler

2.1 Overview

Diderot is an online book system that integrates discussions with content. Diderot consists of two separate systems that are designed to work together:

- the Diderot website, which provides the users (instructors and students) with an online interface for reading books and holding discussions,
- the Diderot compiler, ***DC*** for short, that translates LaTeX and Markdown sources to Diderot-style XML.

Diderot-style XML files may be uploaded onto the Diderot site. In addition to XML, Diderot site accepts conventional PDF documents and slide decks for upload.

In this chapter, we briefly review typesetting for Diderot using LaTex and Markdown. In the next chapter , we discuss some important points about how to publish content on Diderot and how to structure your source code.

Important (Sources and Contributions). To go through the examples in this chapter, please clone this github repository and follow the instructions in INSTALL.md.

2.2 Typesetting with LaTex

DC is compatible with LaTeX. Whatever updates you make to upload your notes on Diderot will remain compatible with LaTeX and you can generate a PDF document from your notes. The reverse direction, going from LaTeX to Diderot, is not as definitive but works reasonably well and with some care you can compile your LaTeX sources to XML.

Definition 2.1 (Books and Booklets)**.** After a LaTeX document is translated to XML via DC, it can be uploaded as a ***chapter*** to a Diderot book or Diderot booklet.

- A Diderot ***book*** is a collection of parts, where each ***part*** is a collection of chapters.
- A Diderot ***booklet*** is a collection of chapters.

Example 2.1. The folders book and booklet contain a sample book (consisting of parts and chapters) and booklet (consisting only of chapters) respectively.

2.2.1 Books, Chapters, and Sections

DC requires a book (or booklet) to be organized into chapters, each of which then contains sections (section, subsection, subsubsection, and paragraph). Each chapter and section can in turn contain "elements" each of which is an "atom" or a "group".

Example 2.2 (Sections)**.** A chapter typically consists of a number of sections.

```
\chapter{Introduction}
\label{ch:introduction}  % Chapters must have a label.

<elements>

\section{A Section}
\label{sec:introduction} % Optional but recommended section label.

<elements>

\subsection{A Subsection}
\label{sec:introduction::sub} % Optional but recommended section label.

<elements>

\subsubsection {A Subsubsection}
\label{sec:introduction::subsub}
<elements>

\paragraph {A paragraph}
\label{sec:introduction::paragraph}
<elements>
```

The term element in the example refers to an "atom" or a "group".

2.2.2 Atoms

Definition 2.2 (Atom)**.** To increase interactivity, Diderot relies on atoms. Atoms are smallest unit of sections and can be included in any section, subsection, subsubsection, and paragraph. In diderot, essentially all content must be inside of atoms. Like any other LaTeX section, an atom cannot be included inside of a LaTeX environment.

An ***atom*** is either

1. a plain paragraph, or
2. a single-standing section of the form

```
\begin{<atom>}[Optional title]
\label{atom-label} % optional but recommended label
<atom body>
\end{<atom>}
```

The term <atom> above can be replaced with any of the following:

- algorithm, answer, assumption,
- code, corollary, costspec,
- datastr (data structure), datatype, definition
- example, exercise,
- gram (non descript atom, i.e., a paragraph),
- hint,
- important,
- lemma,
- note,
- observe, observation,
- preamble (as a first atom of chapter), problem (a problem for students to solve), proof, proposition,
- question, quote,
- remark (an important note), reminder,
- solution (a solution to an exercise/problem), syntax (a piece of syntax)
- task (a task in an assignment), theorem.

Important (Figure and Table Atoms). In addition to the atoms above, Diderot also supports figure and table atoms, which have slightly different syntax:

```
\begin{<atom>}[Optional title]
<atom body>
\caption{Caption}
\label{atom-label} % optional but recommended label
\end{<atom>}
```

Important (Nesting of Atoms). An atom cannot be nested inside another atom and an atom cannot be nested inside a LaTeX environment. This is similar to not being able to define a LaTeX section inside a LaTeX environment.

For example, the following is wrong:

```
\begin{itemize}
\item
\begin{gram}[This is a titled paragraph]
But it is wrong, because it includes an atom inside of a LaTeX environment.
\end{gram}
\end{itemize}
```

Note. Authors can currently use only these atoms defined above. We are working on allowing authors to define their own atoms. In the mean time, you can request new atoms (please send feedback here).

Important. Atoms are ***single standing***, that is to say surrounded by "vertical white spaces" or blank lines on both ends (or LaTeX comments). Therefore, white space matters. In the common case, this goes along with our intuition of how text is organized but is worth keeping in mind. For example, the following code is a plain paragraph atom but not a definition atom, because definition is not single standing.

```
We can now define Kleene closure as follows.
\begin{definition}
...
\end{definition}
```

The following is a definition atom, because it is single standing.

```
\begin{definition}
...
\end{definition}
```

Note. Atoms can contain multiple paragraphs. The following text consists of a text-paragraph atom and a definition atom.

```
Paragraph 0. Sentence 1.
Sentence 2.

\begin{definition}[Definition Title]
Paragraph 1

Paragraph 2
\end{definition}
```

Important (Nesting of atoms).

Controlling granularity

DC will create an atom for each text paragraph. If a piece of text contains many small (one or two line) paragraphs, it can distracting for the user. For example, the following text consists of three small paragraphs.

```
If this then that.

If that then this.

This if and only of if that.
```

We usually don't write like this but sometimes it happens. In such a case, you may want to wrap this text into a single paragraph atom.

```
\begin{gram}[If and only If]
If this then that.

If that then this.

This if and only of if that.
\end{gram}
```

Alternatively you can wrap the text by curly braces as follows.

```
{
If this then that.

If that then this.

This if and only of if that.
}
```

Both will have no impact on the PDF but on Diderot, you will have only one atom for the three sentences.

2.2.3 Groups

Definition 2.3 (Group)**.** A ***group*** consist of a sequence of atoms. DC currently support only one kind of group: flex. On Diderot, a flex will display its first atom and allow the user to reveal the rest of the atoms by using a simple switch. This, for example, can be useful for hiding simple examples for a definition, the solution to an exercise, and a tangential remark, etc.

```
\begin{flex}
\begin{definition}[A Definition]
\label{def:a}

A definition
\end{definition}

\begin{example}[Simple Example I]
\label{ex:a-simple}
Simple example 1
\end{example}

\begin{example}[Simple Example II]
\label{ex:a-simple-2}
Simple example 2
\end{example}

\end{flex}
```

Note. This has turned out be a favorite feature of Diderot for authors and students alike. You can see how this 'flex' example works below, where a definition atom has been paired flexibly with two examples. Click the drawer icon at the bottom left to open the 'flex' and click again to close it.

Definition 2.4 (A Definition)**.** A definition.

Example 2.3 (Simple Example)**.** Simple example 1

Example 2.4 (Simple Example)**.** Simple example 2

2.2.4 Labels and References

Diderot uses labels to identify atoms uniquely. All labels in a book must be unique. To help in authoring, we recommended giving each chapter a unique label, and prepending each label with that of the chapter.

Example 2.5. We recommend labeling content as follows.

```
\chapter{Introduction}
\label{ch:intro}

\begin{preamble}
\label{prml:intro}
...
\end{preamble}

\section{Overview}
\label{sec:intro::overview}


This is a paragraph (atom) without a label.  The following paragraph (atom) has a lal


\begin{gram}
\label{grm:intro::present}
In this  section, we present...
\end{gram}

Here is another paragraph atom, consisting of two environments:
\begin{itemize}
...
\end{itemize}
\begin{enumerate}
...
\end{enumerate}
```

References. To reference a label you can either use

- \href{label}{ref text}
- \ref{label}.

DC replaces the former with \hyperref[][] command so that we can get proper linked refs is latex/ pdf.

DC generates a unique label for each atom, group, and section, in a document. When doing so, DC uses different prefixes for labels: sec for all sections, grp for groups, and the following for atoms. Atoms and their labels are shown below.

```
algorithm : "alg"
assumption : "asm"
code : "cd"
corollary : "crl"
costspec : "cst"
datastr : "dtstr"
datatype : "adt"
definition : "def"
example : "xmpl"
exercise : "xrcs"
hint : "hint"
important : "imp"
lemma : "lem"
note : "nt"
gram : "grm"
observe : "obs"
observation : "obs"
preamble : "prmbl"
problem : "prb"
proof : "prf"
proposition : "prop"
quote : "quo"
remark : "rmrk"
reminder : "rmdr"
slide : "slide"
solution : "sol"
syntax : "syn"
task : "tsk"
theorem : "thm"
```

2.2.5 Code

For code, you can use \lstinline and the lstlisting environment. The language has to be specified first (see below for an example). The Kate language highligting spec should

be included in the "meta" directory and the name of the file should match that of the language. For example if language = C, then the Kate file should be meta/C.xml. If the language is a dialect, then, e.g., language = \{[Cdialect]C}, then the file should be called CdialectC. Kate highlighting definitions for most languages are available online. If you need a custom language, you can probably write one with a bit of effort by starting with the Kate specification for a suitably close language.

Example 2.6 (Python Code)**.** Specify language as python, and include line-numbers on the left of each line. Don't forget to include the Kate highlighting file for python in the meta directory.

```
\begin{lstlisting}[language = python, numbers = left]
def is_even (i):
  if i %2 = 0:
    return true
  else
    return false
\end{lstlisting}
```

The code above will render like this:

```
1 def is_even (i):
2   if i %2 = 0:
3     return true
4   else
5     return false
```

Example 2.7 (Code in C Dialect)**.** Specify the dialect preferred, and don't include line-numbers. Don't forget to include the Kate highlighting file for the dialect in the meta directory.

```
\begin{lstlisting}[language = {[C0]C}]
main () {
  return void
}
\end{lstlisting}
```

2.2.6 Images

You can include JPEG or PNG images by using the usual includegraphics command. We don't support PDF images, because current PDF viewers (browser plugins) are rather clunky and could adversely impact user experience. When including images, there are two points to be careful about: naming and sizing.

When naming the images, it is important the file name of each and every image in a chapter is unique. Diderot indexes images based on their filenames and ignores their paths. For example, it will treat /tmp/image1.jpg and image1.jpg as the same .

When sizing your images, you could, in principle, use absolute sizes, e.g.,

```
\includegraphics[width=5in]{myimage.jpg}
```

The problem with this approach is that the PDF output and the Diderot output will likely have different formats and the image that looks just fine on paper might look too big on Diderot or possibly vice versa. We therefore recommend using relative widths using the following approach

```
\includegraphics[width=0.5\textwidth]{myimage.jpg}
```

DC will translate this to 50% width in html and 50% of \textwidth in PDF and the image will look consistent with its environment in both cases. When specifying the width, it is important to always include the fraction. For example, if you want the width to be exactly equal to \textwidth then use width=1.0\textwidth.

Note. For PDF output, the author could also use \textheight. But this is not meaningful for HTML output, and will not work as expected.

Important (Scale is not Supported). For PDF output it is also possible to use scale. For example,

```
\includegraphics[scale=0.5]{myimage.jpg}
```

This means that the image should be shown at 50% of its actual dimensions. This has the same issues as the absolute measures approach. Furthermore, DC doesn't detect attempt to detect the actual dimensions of the image and this approach is not supported.

Uploading Images. When uploading a chapter, via the UI or via the CLI, the user needs to upload the images explicitly.

2.2.7 Typesetting Mathematics

Diderot uses MathJax to render math environments. This works in many cases, especially for use that is consistent with AMS Math packages. There are a few important caveats.

- Once you switch to math, try to stay in math. You can switch to text mode using \mbox but if you use macros inside the mbox, then they might not work (because mathjax doesn't know about your macros). For example, the following won't work.

  ```
  $\lstinline ‘xyz‘$
  ```

- The "tabular" environment does not work in MathJax. Use "array" instead.

- The environment

  ```
  \begin{alignat}
  ...
  \end{alignat}
  ```

  should be wrapped with \htmlmath, e.g.,

```
\htmlmath{
\begin{alignat}
...
\end{alignat}
}
```

2.2.8 Indexing

In the near term, we plan to index books automatically by doing some natural-language processing. To aid this, we recommend marking definitional terms by using one of the following LaTeX commands: \defb, \defe, \defn. You can define these commands as follows or any other way you prefer.

```
\newcommand{\defb}[1]{\textbf{#1}}
\newcommand{\defe}[1]{\emph{#1}}
\newcommand{\defn}[1]{\textbf{\emph{#1}}}
```

The indexing algorithm will look up these commands and link the use of these terms to their definition.

Example 2.8.

Definition 2.5 (Algorithm)**.** An ***algorithm*** is a recipe for solving a problem.

Definition 2.6 (Single Source Shortest Paths Problem)**.** The ***single source shortest path problem*** or ***SSSP problem*** for short requires finding the shortest paths from a given source vertex to each and every vertex in a graph.

Given these definitions, the following sentence will be processed so that the terms "algorithm" and "SSSP" are linked to the two definitions above. Dijkstra's algorithm solves the SSSP problem.

2.2.9 Colors

You can use colors as follows

```
\textcolor{red}{my text}
```

2.2.10 Embedding Videos

You can embed a video to your notes by using \video{url}{text} command, where text stands for a piece of text that will be displayed on browsers incapable of handling embedded video. For example, the following code, when uploaded on diderot will embed the corresponding YouTube video.

```
Here is a \video{https://www.youtube.com/embed/8QyE9D_3ezQ}{tango video.}
```

Example 2.9 (Video Embedding)**.** The video embedding code will look like this.

Here is a tango video Murat and Michelle Erdemsel. (this video link is not available here)

2.2.11 File Attachments

You can attach a file to a chapter by using the command \href{ file://path.name.ext}{referral text} command. When uploading your chapter, simply include the file name.ext in your attachments.

Example 2.10 (File Attachment)**.** The following code, when uploaded on diderot will create a link to the attached file (requires uploading the file dc.tex as an attachment).

```
Here is the \href{file://dc.tex}{LaTeX source code} for this chapter of the guide.
```

Here is the LaTeX source code for this chapter of the guide.

Important (Uniqueness of Attachments). File name of each and every image and attachment in a chapter most be unique. Diderot indexes images and attachments based on their filenames and ignores their paths. For example, it will treat /tmp/image1.jpg and image1.jpg as the same.

2.2.12 Limitations

LaTeX has grown rich (and perhaps too rich) but many authors tend to use a small subset. DC appears to work well for most uses. Here is a list of known limitations.

- For XML translation work, the chapter should be compileable to PDF. In general, recommended practice is to make sure that you can generate a PDF from your sources and then generate an XML.
- Do not use \input directives in your chapters but you can use them in preambles and packages included at the top of the LaTeX document.
- All macro definitions via newcommand are restricted to the “preamble” of the book.
- Each chapter must have a unique label.
- Fancy packages are not supported. Stick to basic latex and AMS Math packages.
- Support for tabular environment is limited: borders will not show and columns will be centered. You can use the array (math/mathjax) as a substitute, and use \mbox for text fields.
- You can use itemize and enumerate in their basic form. Changing the label format with enumitem package and similar packages do not work. You can imitate these by using heading for your items.

- Labeling and references are limited to chapters, sections, and atoms. You can label these and refer to them, but you cannot label codelines, items in lists, etc.

Remark. The above limitations all pertain to XML generation. Because DC works on a plain LaTeX document, these don't apply to generating other outputs such as PDF.

2.3 Typesetting with Markdown

DC has basic support for Markdown to XML translation.

Header. For translation to work Markdown documents should have either of the following two headers

- Without front matter

```
# Chapter Title
...
```

- With YAML frontmatter.

```
---
Frontmatter
---
# Chapter Title
...
```

Important (Special Symbols and HTML code). We have observed a couple of common mistakes in Markdown that can lead to problems publishing your work on Diderot.

First one concerns special symbols, which should be escaped.

- The hash sign # is a special symbol in LaTeX and must be escaped when using LaTeX macros.
- In math mode % is a LaTeX-Math special symbol and should be escaped.

The second concerns html code. You can use html tags in Markdown for various formatting purposes. If you want to use html syntax for any other reason, then please indicate that it is code. For example, the following is an incorrect use of html code, because it doesn't close the tag table:

```
Here is an example html tag ''<table>''
```

The unclosed html tag here should be placed inside a code, e.g., by using back-tick delimiters.

```
Here is an example html tag '<table>'
```

Important (Chapter Titles). In any given book, all chapter titles must be unique. This ensures that all chapters are uniquely labeled: dc assigns a chapter with title "This is a Chapter Title" the label "ch:this-is-a-chapter-title".

2.3.1 Chapters and Sections

We support the atx-sytle headers, which start by 1-6 hash (#) signs, e.g.,

```
# This is a chapter

## This is a section

### This is a subsection

#### This is a subsubsection

##### This is a paragraph
```

2.3.2 Code

Usual Markdown conventions apply. You can include inline code by using the backtick symbol. For code blocks use three or more backticks and specify the lanugage. It is important to finish a code block with the same number of backtics.

Example 2.11.

````
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8"/>
    <meta content="IE=edge" http-equiv="X-UA-Compatible"/>
    <meta content="width=device-width, initial-scale=1.0" name="viewport"/>
...

```


```ocaml
let x = 2 in
2+2
```


```python
def fun x:
  if x then x
  else x
```
````

2.3.3 Typesetting Mathematics

You can use the usual LaTeX delimiters to include mathematics, i.e., for inline style, use single dollar sign. For math blocks use double-dollars or \[.

2.3.4 Images

You can include JPEG, PNG, and SVG images by using following syntax.

```
![caption](image_file.ext){width=xy%}
```

Example 2.12. The following code displays the image papers.svg.

```
![A categorization of CS papers.](papers.svg){width=90%}
```

2.3.5 Limitations

Markdown to XML translation is relatively basic at this time. If you need a particular feature not supported here, please let us know. Here are few known limitations.

- Markdown usually allows blocks to be ended with equal or greater number of delimiter symbols than the start. For example a codeblock that starts with ‘‘‘ can be ended with ‘‘‘‘‘‘‘ . DC insists on blocks to start and end with the same length delimeters.

- Blank lines in DC will always start an atom. For example if you pepper blank lines into your list, each list item will turn into a separate atom. This is probably not what you have intended.

```
* Item 1 is its own atom.

* Item 2 is its own atom.
```

- DC does not allow nested paragraphs as some Markdown extensions do.

```
* Item 1 is its own atom.

    This is a paragraph that you intended to nest inside Item 1 but
this will start a new atom in DC.

* Item 2 is its own atom.
```

2.3.6 Compiling Markdown To XML

The guide includes a very simple markdown documentmarkdown/example.md.

Making XML. To generate xml, simply use the Makefile provided as follows.

```
$ make markdown/example.xml
```

If you would like to know more about how thing are being handled, you can turn on the verbose option

```
$ make verbose=true markdown/example.xml
```

Using DC directly. You could also use the DC tools directly as follows. The command will generate markdown/example.xml.

```
$ dc markdown/example.md
```

The "Debugger". AS you might notice, dc doesn't currenty give reasonable error messages, partly because if you can generate PDF and html from your LaTeX sources, it is unlikely for there to be errors. In case of an error, you can pass the −d flag to dc to tell her to print the atoms that it matches. See also Section Internals for additional debugging help.

```
dc −d input.md −o output.xml
```

2.4 Internals

When using DC, a high-level understanding of the DC's internal could help. DC works by splitting the LaTeX or Markdown document into atoms, which it treats as units. It uses a Latex to HTML/XML or Latex to HTML/XML compiler to translate each atom and stitches the outputs back to construct the XML for the document. The current implementation works with pandoc , though other compilers such as LaTeXML were also used at some point in the past.

When you receive an error, it will likely look like this:

```
*latex_file_to_html: Executing command: pandoc −−mathjax −−lua−filter ./meta/codebloc
Error in html translation.
Command exited with code: 65
Now exiting.make: *** [dc/dc.xml] Error 65
```

The error occurred when compiling atom # 67, which is stored in the file /tmp/67.tex. To understand the error you can open this file. The file is a complete LaTeX file and you can edit it and run pandoc on it, possible with the flag −−verbose, which will likely be helpful. You can ask pandoc to generate a standalone html, by passing the −s flag, so that you can open the result and examine how it looks rrin your browser.

```
pandoc −−verbose −−mathjax −−lua−filter ./meta/codeblock.lua −−lua−filter ./meta/spar
```

Once you figure out what the problem is what remains is to update your main document and proceed. Basically the same workflow applies to Markdown inputs.

This is all you need to start withe. Enjoy Diderot!

Chapter 3

Publishing Your Work On Diderot

Diderot allows authors to generate interactive books from LaTeX and Markdown sources. Interactive books, however, do raise a number of issues that are not straightforward to handle by starting with just LaTeX/Markdown sources, whose information content makes it difficult to enable certain operations such as live updates. Another difficulty is the complexities of translating LaTeX/Markdown sources (especially LaTeX) to be online friendly.

For ease of use, we therefore strongly recommend the authors to follow certain "best practices." In this chapter, we first describe the basic concepts of publishing on Diderot and then describe the structure of a Diderot "book" and a "booklet" and our recommendations for organizing these. For brevity, we consider LaTeX sources here, but the practices generalize to Markdown sources.

Important (Sources). To go through the examples in this chapter, please clone this github repository and follow the instructions in INSTALL.md. The examples are tested on Mac OS X and Ubuntu. The binaries in bin might not work on systems that are not Mac or Linux/Unix-like.

3.1 Released and Unreleased Content

You can publish content on Diderot in two different forms: "unreleased" and "released". When you first upload some content in the form of a book chapter on Diderot, the chapter is unreleased by default. You can release the chapter by using Book Managemement features, but before you do, beware that releasing should be treated as a one-way feature.

Released Content: Implications for Authors and Users. After you release a chapter, it becomes available to your readers, who might start taking notes on its content and discussing it, by for example asking questions to course staff. In other words, released content is out in the wild and has now affected users.

Updating content after it is released therefore raises a number of questions, e.g.,

- should it be allowed at all? (For example, fixing an error pointed out by a student makes a good question look like an unnecessary question.)
- if it should be allowed, then under what circumstances should it be allowed, who should decide?

Technical Issues with Live Updates. Beyond these important questions, live-updates to released content are complex from a technical point of view. The first issue is identifying what has been changed and what has not. To see why let's say that you have published and released a chapter on Diderot and later realized that you want to make some updates, and so you edited the chapter sources. Now, if you want to update the released chapter on Diderot, then Diderot needs to figure out which sections and atoms have been edited and which ones remain the same. This is not easy, because Diderot doesn't know the full history of your edits.

A second technical concern is treatment of deleted content. Let's say that you have deleted a paragraph, but a student took personal notes on this paragraph. What should happen to students' notes? Should they also be deleted? If not, then they would be linked with deleted content, effectively making their notes unusable.

Example 3.1. Let's say you have edited a released chapter and swapped the position of two paragraphs. This change is ambiguous to Diderot, because it cannot distinguish this edit from complete rewriting of the two paragraphs (i.e., did you swap or did you rewrite?). The distinction is important, because it impacts how the updates affects the users.

Releasing Content is One Way. These challenges are not insurmountable but dealing with them in the "right way" requires the development of a rather complex infrastructure to resolve with conflicts created by live updates. At this time, we have therefore decided to proceed with this principle: a chapter will not be updated after it is released.

Specifically, Diderot refuses to live update a (released) chapter if it *"deletes live blocks"*, which means that proceeding with the update deletes a block (an atom, a group, or a section), which has been released to the users.

There is one important exception to this rule, which allows the authors to perform live updates reasonably liberally. We discuss this approach, which is based on "atomic sources" in the next section .

Best Practices. The best practice for publishing content on Diderot is to publish it

- in unreleased form first,
- inspect the material,
- ideally, have your fellow co-instructor and TAs give you feedback (Diderot allows sending feedback on unreleased content), and
- update the content according to feedback, and then release it.

We have designed this process to facilitate easy updates for unreleased content. Course staff may give feedback on unreleased chapters, and upon updates, such feedback will be deleted, eliminating thus otherwise inconsistent feedback from released chapters.

Live Update. Mistakes do happen and there are several ways to fix errors in released chapters. If your sources are atomic, live updates are generally feasible . If your sources are not atomic, then the following options are available.

- **Unreleasing:** In some circumstances, you can simply ***unrelease*** a chapter, effectively pulling it from user's view (excepting your course staff). For example, if you accidentally released a chapter and a small amount of time has passed, you might choose to unrelease a chapter so that you can make some updates. Or if you released a chapter, and noticed that it is corrupt in a big way, you might choose to unrelease the chapter to prevent confusion among the users.

  Once unreleased, you can update the chapters as you wish, and all other user interactions prior to the updates will be deleted. After you make your updates, you can release the chapter as usual. Because unreleasing deletes all student interactions, this option should be used with care.

- **Online edits:** for small corrections, perhaps the easiest way is to edit the affected atoms online on Diderot. For example, this is usually sufficient to fix small typos. (Don't forget to update your LaTeX or Markdown source).

- **XML edits:** For larger changes, you can edit the XML of the chapter directly and reupload the XML. When editing the XML, it is very important not to alter label fields and update only the body and title fields.

- **LaTeX edits:** Certain LaTeX and Markdown edits are also feasible. For example, you can simply add new content to the tail of your LaTeX/Markdown chapter source but as you do this, it is very important to not alter any of the prior content. Similarly, if you want to change your preamble file (where your macros are defined), then you will be able to use your LaTeX sources to generate and XML and re-upload the file.

- If none of the above work, you can "abandon" your released chapter. Let's say that you released Chapter 2 a while back and noticed later some serious problems with it. You don't feel comfortable to unrelease it because it has been a week since its release, the above solutions don't apply. In this case, you can deprecate this chapter by changing its title to "Deprecated" and creating a Chapter 2.1 (Yes, chapters can be assigned real numbers, where the decimal is thought as a version number) with the updated content.

The Innerworkings. It might help to understand the innerworkings of Diderot a bit to understand the limitations on live updates. When you use Diderot's compiler to compile your sources to XML, the compiler generates a unique ***label*** for each block (atom, group, or section). If the author has given a label to the block, then Diderot uses that label. Otherwise, it generates one. At a very high level, the labels are generated by using the text itself. For example, if you have a definition that defines a "graph", Diderot's compiler might label it

as def:graph−theory::graph, where graph−theory is the label of the chapter, and the word graph is pulled from the text of your definition.

Diderot uses labels as identifiers of all content. For example, if a student asks a question about a definition atom, a link between the student and the definition (as identified by its label) is established. Likewise, if you update a released chapter, Diderot matches blocks based on their labels and updates the matching blocks in place.

Now, changing your latex sources could change the labels auto-generated by Diderot's compiler. This in turn will lead to the upload deleting a block that is unmatched. Therefore, simply giving a label to each of your atoms allows you to perform updates on released content. This is the basic idea behind the atomic method .

3.2 The Atomic Method for LaTeX

If you use LaTeX, then there is a way to write your sources in such that many kinds of live updates are feasible on released content. The idea is to "atomize" your chapter fully by making sure that each piece of text is inside of an atom and giving each section, group, and each atom a unique label and never change these labels (at least if you intend to live update).

Example 3.2 (An Atomic Assignment)**.** Here is a fully atomic assignment. Note that the chapter, all sections, and all atoms have labels. When assigning labels, note that they must be unique within the book; it is therefore good practice to prefix them with chapter label.

```
\chapter{Assignment 1}
\label{ch:asg1}

\begin{preamble}
\label{prml:asg1::intro}
In this assignment, you will use your knowledge about complexity classes..
\end{preamble}

\section{The Class of P}
\label{sec:asg1::p}

\begin{problem}[String Matching in P]
\label{prb:asg1::p:string-matching}
Prove that an algorithm that checks that two string are equivalent are in P.
\end{problem}


\section{The Class of NP}
\label{sec:asg1::np}

\begin{problem}[String Matching in NP]
\label{prb:asg1::np:string-matching}
Prove that an algorithm that checks that two string are equivalent are in NP.
```

```
\end{problem}


\section{P versus NP}
\label{sec:asg1::p-versus-np}

\begin{problem}[P versus NP]
\label{prb:asg1::p-vs-np::p-vs-np}
Prove that if P = NP, then life is good.
\end{problem}
```

Important. The atomic approach is usually feasible for small texts such as assignments but tends to be too labor intensive for larger text such as book chapters. We have a compilation tool that can automatically atomize chapters for you but this tool is currently in experimental stage, if you are interested in experimenting with it please contact Umut .

Labeling all blocks in a chapter allows Diderot to match them when you want to update a released section: blocks with the same label are matched and the online content is updated with the new one. This means that

- you can edit the text within the atoms as you wish,
- you can add new labeled blocks as long as you don't change the labels of existing ones, e.g., you can add new problems
- you can "delete" content by leaving the atom in and delete its text.

If you make such updates, especially that changes or removes existing content, please be mindful of their implications of live updates for the users .

Note. You can read more about labels and references here.

3.3 Structuring your Sources

We recommend following the structure of the book and the booklet examples included in this guide source code (in folders 'book' and 'booklet') and described here. As examples, the folders book and booklet in guide sources contain several files.

- book.tex

  This is the top level file for PDF generation. It is a conventional LaTeX document is but is organized carefully for Diderot compatibility.

- book−html.tex

  This is the top level file for html generation. It is mostly a copy of book.tex but it uses slightly different packages.

- templates/diderot.sty

  Supplies diderot definitions needed for compiling latex to pdf's. You don't need to modify this file.

- templates/packages.sty

  Supplies packages for PDF generation. You can add whatever package you want. Most of these packages, except perhaps basics ones and AMS Math packages, will be ignored for XML generation. It is important not to include any command definitions in this file.

- templates/preamble.tex

  Supplies your macros that will be used by generating a PDF via pdflatex. All macros should be included here and no packages should be included (use packages.sty for packages.

- templates/preamble−diderot.tex

  Equivalent of preamble.tex but it is customized for XML output. This usually means that most macros will remain the same but some will be simplified to work with 'pandoc'. If you don't need to extensive customization, you can keep just one preamble.tex and input that file in your preamble−diderot.tex, as in the booklet example.

```
% File: preamble-diderot.tex

% Includes all of preamble.tex
\input{./templates/preamble.tex}

% Redefine \mybold command defined in preamble.tex for Diderot.
\renewcommand{\mybold}[1]{\textbf{#1}}
```

Important. The idea behind organizing preamble−diderot.tex by including preamble.tex and modifying it as needed is to minimize human errors that can be more difficult to fix once published on Diderot. We strongly recommend the authors to follow this practice.

To simplify publishing on Diderot, we recommend organizing your booklet and book sources as outlined below.

Booklets. Booklets are books that don't have parts. We recommend creating one directory per chapter and placing a single main.tex (a chapter specific name could also be used) file to include all contain that you want. Place all media (images, videos etc) under a media/ subdirectory.

- ch1/main.tex or ch1/ch1.tex
- ch1/media/: all my media files, *.png *.jgp, *.graffle, etc.
- ch2/main.tex or ch2/ch2.tex
- ch2/media/: all my media files for chapter 2, *.png *.jgp, *.graffle, etc.
- ch3/main.tex or ch3/ch3.tex

Books. Books have parts and chapters. We recommend structuring these as follows, where 'ch1, ch2' etc can be replaced with names of your choice.

- part1/ch1.tex
- part1/ch2.tex
- part1/media−ch1/
- part1/media−ch2/
- part2/ch3.tex
- part2/ch4.tex
- part2/ch5.tex
- part2/media−ch3/
- part2/media−ch4/
- part2/media−ch5/

Making PDF and HTML. Inside the book\ and booklet\ directories, you can use pdflatex to generate a PDF output. See the corresponding Makefile. For example, you can invoke the Makefile as follows to make a PDF:

```
$ make pdf
$ make html
```

Making PDF/HTML for a Single Chapter. To make specific chapters, extend the Makefile to compile each chapter separately. See the Makefile in book or booklet as examples. For example, you can compile the chapter probability in the book directory as follows.

```
$ make probability
```

To generate HTML for a single chapter, update book−html.tex to exclude all other chapters and run make html.

Making XML. XML generation proceeds on a chapter by chapter basis. To generate xml, simply use the Makefile provided as follows.

```
$ make ch2/main.xml
```

To obtain more information from a run, you can turn on the verbose option.

```
$ make verbose=true ch2/main.xml
```

Beware that you will likely see many warnings. For this reason the utility of the verbose flag is not particularly high.

Important (PDF before XML). Before generating the XML for a chapter, please check first that you can generate PDF for the chapter. This is because XML generation is relatively "forgiving" and, for example, simply omits undefined macros without complaining. By generating PDF first, the author could avoid many errors that can be more time-consuming to fix after the material is published on Diderot.

Note. The "PDF before XML"

Error messages from the XML translator are not particularly useful at this time. But, if you are able to generate a PDF and HTML (using pandoc), then you should be able to generate an XML. If you encounter a puzzling error try the "debug" version which will print the atoms as it goes through the input.

```
$ make debug=true probability/theory.xml
```

If that fails, there is another relatively simply way to debug, see Section Internals for additional information.

Using DC directly. Assuming that you structure your book as suggested above, then you will mostly be using the Makefile but you could also use the DC tools directly. This tools translates the given input LaTeX file to xml.

```
$ dc --meta ./meta --preamble preamble.tex input.tex -o output.xml
```

To turn on the debug mode, simply pass the flag -d.

```
$ dc -d --meta ./meta --preamble preamble.tex input.tex -o output.xml
```

The meta folder contains some files that may be used in the XML translation. You can ignore this directory to start with and then start populating it based on your needs. The main file that you might want to add are Kate highlighting specifications to be used for highlighting code.

Chapter 4

Diderot Command Line Interface

Diderot has a command line interface available to make interacting with the platform a little easier if you prefer using the command line instead of web UI's.

Important (Sources and Contributions). To download the CLI, please clone this github repository and follow the instructions in README.md.

The Diderot CLI is public. Feel free to contribute or open an issue if there are any commands or utilities that you think would be useful to have. Please send feedback about the CLI on Diderot.

Note (Questions and Feedback). You can click any "mail" button in this guide and send feedback or ask questions.

4.1 Basic Usage

Use the `diderot_student` or `diderot_admin` scripts to interface with the CLI. For help about the CLI, use the `-h` or `--help` commands.

4.1.1 Credential Management

Credentials are passed to the CLI in one of two ways. The first is to simply pass your credentials via the CLI using the `--username` and `--password` flags on the CLI. A more convenient (and recommended) way is to use a *credentials file*. The credential file format is simply a text file with your username on the first line and your password on the second. Use the `--credentials` argument to point the CLI towards a file containing your credentials. For easier usage, the CLI automatically looks at the files `~/private/.diderot/credentials` and `~/.diderot/credentials` for a credentials file of this form. If this file exists, then the CLI will automatically log you in, and no credentials need to be explicitly provided to the CLI. An important note is that your credentials file must have only "owner can read

and write" permissions. To do this, run `chmod 600 <credentials file>`. An example credential file is below.

```
MyUsername
MyPassword
```

Note. The `--username/--password` or `--credentials` arguments must be passed to the CLI before the specific commands below!

For example, this command is correct:

```
diderot_student --credentials Path/To/My/Creds list_courses
```

This command is incorrect:

```
diderot_student list_courses --credentials Path/To/My/Creds
```

4.2 Student Interface

- `list_courses`: List all available course labels that the user can access. Operations which require a course input must use the one of the labels returned from this command.
- `list_assignments`: Lists all assignments for a course.
- `download_assignment`: Downloads all handout/writeup files (if present) from a target course and assignment into the current directory.
- `submit_assignment`: Submit a handin.tar file to a given course and homework.

More details about each command are below.

4.2.1 list_courses

`list_courses` lists all course labels that the user has access to. The labels returned from `list_courses` must be used as arguments to all other operations that require a course label. It accepts 0 arguments.

Example Usage:

```
diderot_student list_courses
```

4.2.2 list_assignments

`list_assignments` is used to list all available assignments for a particular course. It accepts only 1 argument.

- `course`: Course label to list homeworks of.

Example Usage:

```
diderot_student list_assignments courselabel
```

4.2.3 download_assignment

`download_assignment` is used to download handout files associated with an assignment. It attempts to download all handout files into the current directory. It accepts 2 arguments.

- `course`: Course label that the homework belongs to.
- `homework`: Name of the homework to download handouts of.

Example Usage:

```
diderot_student download_assignment courselabel homeworkname
```

4.2.4 submit_assignment

`submit_assignment` is used to submit handin files to an assignment. As of now, it **only** supports uploading handin tar files. If your assignment is configured to accept anything other than tar files, do not use this method to submit. It accepts 3 arguments.

- `course`: Course label that the homework belongs to.
- `homework`: Name of the homework to submit to.
- `handin_path`: Path to the handin.tar file.

Example Usage:

```
diderot_student submit_assignment courselabel homeworkname ~/Path/To/My/Handin
```

4.3 Admin Interface

The admin CLI contains all the above commands, and expands the set to include some commands relating to manipulation of books and assignments.

- `list_books`: List all books, can optionally filter by course.
- `list_parts`: List all parts of a book.
- `list_chapters`: List all chapters of a book.
- `create_part`: List all parts of a book.
- `create_chapter`: Create a chapter in a given part of a book. Optionally specify the new chapter's label and or title. If not, these values are filled in with a default.
- `upload_chapter`: Update the content of an existing chapter. Upload either a standalone pdf, pdf of a slideshow, or an xml document and image attachments. The image upload accepts filenames, globs, and folder names (folders are recursively traversed).
- `update_assignment`: Update the files relating to an existing homework on Diderot. Upload any of the autograder tar, autograder makefile, pdf writeup, and student handout.

4.3.1 list_books

`list_books` is used to list book labels that the user has access to. The result from `list_books` is used as the argument book label for commands that require a book label. It accepts 2 arguments.

- `course`: Course label that the target book belongs to.
- `--all`: Optional flag to list all books regardless of course. Not to be used when specifying a course.

Example Usage:

```
diderot_admin list_books courselabel
diderot_admin list_books --all
```

4.3.2 list_parts

`list_parts` is used list the parts and numbers within a target book. It accepts 2 arguments.

- `course`: Course label that the target book belongs to.
- `book`: Label of the target book.

Example Usage:

```
diderot_admin list_parts courselabel booklabel
```

4.3.3 list_chapters

`list_chapters` is used list the chapters and numbers within a target book. It accepts 2 arguments.

- `course`: Course label that the target book belongs to.
- `book`: Label of the target book.

Example Usage:

```
diderot_admin list_chapters courselabel booklabel
```

4.3.4 create_part

`create_part` is used to create a part within a target book. It accepts 5 arguments.

- `course`: Course label that the target book belongs to.
- `book`: Label of the target book.
- `title`: Title to give to the newly created part.
- `number`: Number to give the newly created part.
- `--label`: Optional label to give to the newly created part. It defaults to a random value if not provided.

The command will error out if there already exists a part within the book with the same number.

Example Usage:

```
diderot_admin create_part courselabel booklabel title number --label newlabel
```

4.3.5 create_chapter

`create_chapter` is used to create a chapter within a target book. It accepts 6 arguments.

- `course`: Course label that the target book belongs to.
- `book`: Label of the target book.
- `--part`: Number of the part to create the chapter within. This argument is not used when the target book *is a booklet*.
- `--number`: Number to give the newly created chapter.
- `--title`: Optional title to give to the newly created chapter. It defaults to 'Chapter'.
- `--label`: Optional label to give to the newly created chapter. It defaults to a random value if not provided.

The command will error out if there already exists a chapter within the book with the same number.

Example Usage:

```
diderot_admin create_chapter courselabel booklabel --part partnum \
--number newnum --title newtitle --label newlabel
```

4.3.6 upload_chapter

`upload_chapter` is used to upload PDF, slide, or xml/mlx content to a chapter. It accepts quite a few arguments and combinations.

- `course`: Course label to upload chapter to.
- `book`: Book label to upload a chapter to.
- `--chapter_number` or `--chapter_label`: Use one of these arguments to specify which chapter to upload content to. Use either the chapter's number within the book, or its label.
- `--pdf` or `--slides` or `--xml`: Use one of these arguments, specifying the desired file type to upload. Note that your slides must be in a PDF format to use the `--slides` argument. The `--xml` argument accepts both XML and MLX file formats.
- `--video_url`: Optional URL to a video to embed within the chapter. This argument can only be used with the `--pdf` and `--slides` arguments.
- `--xml_pdf`: Optional path to a PDF version of the XML content for printing. This argument can only be used with the `--xml` argument.

- `--attach`: Optional list of paths to attachments to upload along with an XML chapter upload. This argument can only be used with the `--xml` argument. The command accepts paths to individual files, globs, and folders. Folders are recursively traversed.

Example Usage:

```
diderot_admin upload_content courselabel bookname chapternumber \
--pdf Path/to/Pdf --video_url someurl

diderot_admin upload_content courselabel bookname chapternumber \
--slides Path/to/slides --video_url someurl

diderot_admin upload_content courselabel bookname chapternumber \
--xml Path/to/Xml --xml_pdf Path/To/XmlPdf \
--attach Path/To/File Path/To/Folder/ Path/With/Glob/*
```

4.3.7 update_assignment

`update_assignment` is used to update files related to an assignment on Diderot. It accepts 6 arguments.

- `course`: Course label that the assignment belongs to.
- `homework`: Name of the assignment to update.
- `--autograde-tar`: Optional path to the autograder tar file for the assignment. The autograde.tar file contains what is needed to run the assignment's autograder.
- `--autograde-makefile`: Optional path to the autograde-Makefile for the assignment. The autograde-Makefile is used to execute the autograder on a student handin.
- `--writeup`: Optional path to a PDF writeup of the assignment.
- `--handout`: Optional path to a handout file that students can download to start the assignment.

Example Usage:

```
didetot_admin update_assignment courselabel homeworkname \
--autograde-tar Path/To/Tar \
--autograde-makefile Path/To/Makefile \
--writeup Path/To/Writeup --handout Path/To/Handout
```