Overview

Diderot is an online book system that integrates discussions with content. Diderot consists of two largely separate systems that are designed to work together. The first is the Diderot site, which provides the users (instructors and students) with an online interface for reading books and discussions. The second is the MTL (read "metal") compiler that translates LaTeX and Markdown sources to XML, which can then be uploaded onto the Diderot site. In addition to XML, Diderot site accepts conventional PDF documents and slide decks for upload. This document describes MTL and its use.

For any questions or comments, please don't hesitate to contact Umut at `umut@cs.cmu.edu`.

Typesetting with LaTex and MTL (MeTaL)

MTL tries to remain compatible with LaTeX. If you have LaTeX sources that you are able to compile and generate PDF from, then in most cases, you can use MTL to generate XML from your LaTeX sources. Translation of LaTeX sources to XML is not perfect at this time but works quite well for simple LaTeX sources.

Examples

See directories `book` (a book with parts and chapters) and `booklet` (with chapters and no parts) for examples diderot books and chapters. These are set up with Makefiles so that you can generate both PDF and XML from the sources. For each book you generate the book PDF by running the following.

```
$ make book.pdf
```

You can similarly generate the XML for each chapter, which can then be uploaded onto diderot.

```
$ make graph-contraction/star.xml
```

Perhaps the most important requirement is this: all packages and macros that your book relies on should be placed into a single "preamble" file. See "Compilation" section below for more details.

Basic syntax

Segments

MTL thinks of a LaTeX source as being organized in terms of segments (chapters, sections, subsections, subsubsections, paragraphs). Each segment in turn consists of *elements* which are *atoms* or *groups*.

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

\begin{preamble}
\label{intro::preamble} % Optional but recommended atom label.
 ...
\end{preamble}

\section{Overview}
\label{sec:intro::overview} % Optional but recommended section label.
<elements>

\subsection{An Example}
\label{sec:intro::overview::example} % Optional but recommended section label.

<elements>

\subsubsection {...}
\label...
<elements>

\paragraph {...}
<elements>
```

Here `elements` is a sequence of "atoms" and "flex's".

Atoms

An *element* is an atom or a group.

An *atom* is either 1) a plain paragraph, or 2) a single-standing environment of the form

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

Note that atoms are defined by "vertical white spaces", i.e., they are single standing and are not surrounded by other text. White space therefore matters. In the common case, this goes along with our intuition of how text is organized but is worth keeping in mind.

In addition to plain paragraphs, there are many atoms to choose from. Here is a complete list. Let me (umut@cs.cmu.edu) know if you want others.

- `algorithm`
- `assumption`
- `code`
- `corollary`
- `costspec`
- `datastr`
- `datatype`
- `definition`
- `example`
- `exercise`
- `hint`
- `important`
- `lemma`
- `note`
- `gram` (non descript atom, i.e., a paragraph)
- `preamble` (only as the first atom of chapter)
- `problem`
- `proof`
- `proposition`
- `remark`
- `reminder`
- `solution`
- `syntax`
- `task`
- `teachask`
- `teachnote`
- `theorem`

Currently, we only allow you to use these atoms. You can also ask Umut to create new atoms if you need one.

Groups

A *group* consist of a sequence of atoms. We 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. We find `flex` groups to be useful for hiding simple examples for a definition, the solution to an exercise, and sometimes tangential remarks.

```
\begin{flex}
\begin{<atom>}[optional title]
\label{atom-label}


\end{<atom>}

\begin{<atom>}[optional title]
\label{atom-label}


\end{<atom>}

<... additional atoms if desired>
\end{flex}
```

Labels

Labels play an important role in Diderot, because they allow identifying atoms uniquely. It is a good practice to try to give a label to each atom, flex, section, subsection...

Important: All labels in a book must be unique. Diderot generates labels for all atoms even if you don't give them one; see the tool `texel`.

I recommended giving each chapter a unique label, and prepending each label with that of the chapter, e.g.,

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

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

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


Here is a paragraph atom without a label.


\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}
```

The following label prefixes are recommended. You can ask MTL to generate labels for you, using the `texel` tool. In doing so, MTL will use the following prefixes for each kind of atom

```
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"
preamble : "prmbl"
problem : "prb"
proof : "prf"
proposition : "prop"
remark : "rmrk"
reminder : "rmdr"
slide : "slide"
solution : "sol"
syntax : "syn"
task : "tsk"
theorem : "thm"
```

Code

For code, you can use 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.

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

Label references

Use

`\href{label}{ref text}`

for references or the standard

`\ref{label}.`

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

Colors

You can use colors as follows

`\textcolor{red}{my text}`

Code

Use `lstinline` and always specify the language as first option

Example:

```
\begin{lstinline}[language=C, numbers=left]
...
\end{lstinline}
```

```
\begin{lstinline}[language={[CO]C}, numbers=left]
...
\end{lstinline}
```

Limitations

- For XML translation work, the chapter should be compileable to PDF.
- Do not use \input directives in your chapters.
- Each chapter must have a unique label.
- Fancy packages will not work. Stick to basic latex and AMS Math packages.
- Support for tabular environment is limited: borders don't work, neither does columnt alignment, columns are centered. You can use the array (math/mathjax) as a substitute. This could require using for text fields.
- Center environment doesn't work.
- For figures specify the width/height in terms of concrete units, e.g., width = 4in, height = 8cm.
- You can use itemize and enumerate in their basic form. Changing label format with enumitem package and similar packages do not work. You can imitate these by using heading for your items.
- In general labeling and referencing is relatively limited to atoms. You can label atoms and refer to them, but you cannot label codelines, items in lists, etc.
- We use mathjax to math environments. This works in many cases, especially for AMS Math consistent usages. There are a few important caveats.
  - Once you switch to math, try to stay in math. You can switch to text mode using but if you use macros inside mbox, they might not work (because mathjax don't know about your macros). For example, this won't work

```
$\lstinline'xyz'$
```

this should be outside math.

  - 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}
```

```
}
```

Compilation

The following instructions 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.

Overview

See as examples the directories `book` and `booklet`.

The relevant files are

- `templates/diderot.sty`

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

- `templates/preamble.tex`

  Supplies your macros that will be used by generating a pdf via pdflatex. Nearly all packages and macros should be included here. Each chapter will be compiled in the context of this file. Ideally this file should

  - include as few packages as possible
  - define no environment definitions
  - macros should be simple

- `templates/preamble-mtl.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 customize, you can keep just one preamble. The example in directory `booklet` does so.

Structuring your books sources

I recommend structuring your book sources in a way that streamlines your workflow for PDF generation and Diderot uploads. I have found that the structure outlined below separately for booklets and books work well. The example book and booklet provided follow this structure (see directories `book` and `booklet`).

Booklets

Booklets are books that don't have parts. For these I recommend creating one directory per chapter and placing a single main.tex file to include all contain that you want. Don't use \input's within the tex files. Place all media (images, videos etc) under a media/ subdirectory.

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

Books

Books have parts and chapters. I 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 of the whole book or booklet

```
$ make book.pdf
```

Making PDF a specific chapter

- Extend book.tex to include the chapter
- Extend Makefile, follow example.

To compile ch2 type

```
$ make ch2
```

Making XML of a specific chapter

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

Error messages from the XML translator are not useful. But, if you are able to generate a PDF, then you should be able to generate an XML. If you encounter a puzzling error try the "debug" version which will give you an idea of where it blew up.

```
$ make ch2/main.xmldbg
```

Usage

Assuming that you structure your book as suggested above, then you will mostly be using the Makefile but you could also use the MTL tools directly.

Tool: texml

This tools translates the given input LaTeX file to xml.

Example: `texml -meta ./meta -preamble preamble.tex input_file.tex -o output_file.xml`

The meta direcotry 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.

Tool: texml.dbg

This tools is the "debug" version of the texml binary above. As you might notite, `texml` doesn't currenty give reasonable error messages. The debug version prints out the text that it parses, so you can have some sense of where things have gone wrong. As you will likely experience, `texml` should work if your latex sources are otherwise correct (you can run them through pdflatex), so hopefully, you will not have to use this binary much.

Example: `texml -meta ./meta -preamble preamble.tex input_file.tex -o output_file.xml`

Tool: tex2tex

This tools reads in your LaTeX sources, parses them, and writes it back. It drops comments and normalized the whitespace but should otherwise return

back a LaTeX file that is essentially the same as the input file. You should not need to use this binary, which is primarily used for testing during development.

Examples:

```
$ bin/tex2tex ./graph-contraction/star.tex -o ./s.tex
$ diff ./graph-contraction/star.tex ./s.tex
```

Tool: texel

This tool "normalizes" your latex sources. This means that it

- atomizes your code, wrapping each paragraph into a non-descript "gram" atom if it is not already wrapped.
- wraps each atom by a "group", if not already wrapped by one.
- gives each segment (section, subsection, subsubsection, paragraph, atom) of the input file a label and it wraps each atom into a "group" if it is not already in a group. A group is one of "cluster" "flex" "mproblem" (multipart problem).

Generated labels have the form

`kind_prefix:chapter_label:segment_label`

Here kind_prefix could for exmaple be * `sec`, for section, subsection, subsubsection, paragraph * `xmpl`, `thm`, for an example or a theorem.

The chapter_label is extracted from the chapter label given. For exmaple, if the label has any one of the form

`ch:star | chapter:star | ch_star | ch__star | ch:_star | chap:_star`

chapter_label will be `star`.

The tool takes the label, split it at the delimiters [:_]+ and if the prefix starts with "ch" it take the rest of the label as the chapter label.

Some example full labels: * xmpl:star:simpleexample * thm:star:costbound

Typesetting with Markdown and MTL (MeTaL)