Modular Casebook Management: modbook.sty

Charles Duan
cduan@wcl.american.edu

Version v1.0.0, 2024/01/09

Contents

1 Introduction

This is a package for managing the compilation of a textbook made up of several interdependent modules. The purpose of this package is:

- To manage cross-dependencies between parts of the textbook. For example, one part may reference a case that should have been already read in the book, so it should be possible to raise a warning if that case hasn't already been included.
- To provide formatting for standard parts of a casebook.
- To permit local alterations to casebook files. This requires devising a directory structure for local files, such that including a file searches first for the local copy and then for default version.

The general workflow model assumed by this package is as follows. Textbooks are to be compiled out of cases, articles, and other materials described in this documentation as *readings*. One or more *editors* compile, edit, and annotate these readings, and perhaps write editorial material of their own. The editors arrange their work into *modules*, each of which contains an outline and content files including readings. A *compiler* then receives modules and arranges them into a book. The compiler may also wish to make changes to the editors' work.

2 Installation and Dependencies

The package uses hyperref for internal cross-references, and hicite for URL formatting and a few other things. It also requires the graphicx and etoolbox packages.

Because hicite is included, you may use it for managing citations within the casebook as well, although this is optional.

3 Modules

This package uses a rigorous file hierarchy to manage the work of editors and compilers. At the base of the hierarchy are *repositories* of content. Generally an editor (or team of editors) would be responsible for a single repository. Within each repository are *modules* contained in subfolders.

3.1 Directory Structure

Modules contain content files consistent with the following rules:

- ⟨*module*⟩`/`⟨*module*⟩`.tex`: The default outline of the module. This file should contain only section headings, `question` environments, and `\import` commands.
- ⟨*module*⟩`/intro-`⟨*module*⟩`.tex`: The introductory text for a module should by convention have this name, making it convenient to determine whether a module has been imported into a book.[1]
- ⟨*module*⟩`/intro-`⟨*filename*⟩`.tex`: A file containing editorial or introductory text (that is, text not part of a case or other reading).
- ⟨*module*⟩`/narrative-`⟨*filename*⟩`.tex`: The same as an `intro-` file. (Starting a filename with `narrative-` can be used to indicate that the text is intended as a full standalone section, rather than as an introduction to another reading.)
- ⟨*module*⟩`/`⟨*filename*⟩`-qs.tex`: A list of questions and notes that may follow a reading. By convention, ⟨*filename*⟩ corresponds to the file to which the questions apply. A question file will be included inside a list-like environment, so items should begin with `\item`.
- ⟨*module*⟩`/`⟨*filename*⟩`.tex`: Any other filename is assumed to be a reading from an external source (which should start with `\reading`).

`\RepositoryPath` `{`⟨*path,...*⟩`}`

Provides a list of repository directories. The package will sequentially search through each repository given in the argument, which should be a comma-separated list, until it finds the module file required. The default path is `local,base`.

3.2 Importing Modules

`\import` ⟨*name*⟩

The `\import` command is the key command for incorporating content files into a casebook. It is used both in the overall book to import modules, and in the module files (primarily ⟨*module*⟩/⟨*module*⟩.tex) to import content for each module.

[1] It is generally inadvisable to check if the ⟨*module*⟩`/`⟨*module*⟩`.tex` file itself has been imported, because compilers will often not use the default outline when selecting parts of modules.

The macro takes one argument, which need not be surrounded by braces, similar to `\input`. The argument may be:

1. A content file without a module name. The file is assumed to be within the last-`\import`ed module.
2. A content file with a module name (⟨*module*⟩`/`⟨*filename*⟩).
3. A module name alone, in which case ⟨*module*⟩`/`⟨*module*⟩`.tex` is used.

3.3 Cross-Reference Expectations

Content in modules will often cross-reference material in other modules. But if the compiler can select and reorder the modules, these cross-references will become unanchored. The package thus provides several macros to manage cross-references. Editors should insert these macros into their module files as they write, enabling their modules and files to be rearranged without creating contextual problems.

Expectations are defined based on filenames, and are met if a corresponding file has been `\import`ed into the book at the correct time. Filenames may be given with the module name or without. (The best practice, then, is to ensure that filenames are unique even across different modules.)

Failures of any of the expectation assertions below will result in a warning and an undefined-reference warning at the end of document compilation.

`\having` `{`⟨*filename*⟩`}` `{`⟨*before*⟩`}` `{`⟨*after*⟩`}` `{`⟨*none*⟩`}`

Chooses a text depending on the inclusion status of ⟨*filename*⟩. If the file has already been `\import`ed, then ⟨*before*⟩ is used. If the file is imported later, then ⟨*after*⟩ is used. If the file is never imported, then ⟨*none*⟩ is used.

This macro best enables flexibility for compilers, and should be used in preference to the other expectation assertion macros to the extent possible.

`\expected` `{`⟨*filename*⟩`}`

Tests whether a file has been included already, and produces a warning if not.

`\expecting` `{`⟨*filename*⟩`}`

Tests whether a file will later be included. The test fails if the file is never included, or if the file was included before this command was called. Because it relies on the `.aux` file, this command may produce spurious warnings that go away on subsequent compilations.

`\expectnext` {⟨*filename*⟩}

Indicates that the next imported file should match this filename. This is used, for example, at the end of an introductory text intended to precede a reading.

4 Formatting Content

Casebooks generally use only a few types of materials for readings, and also include common types of editorial content. The macros here help with formatting these elements consistently.

4.1 Readings

These commands are useful for formatting a reading from a case or other materials. Typical usage is as follows:

```
\readingnote{Decided on the same day as Bolling v. Sharpe, 347
U.S. 497 (1954).}
\reading{Brown v. Board of Education}
\readingcite{347 U.S. 483 (1954)}

\opinion \textsc{Mr. Chief Justice Warren} delivered the opinion
of the Court.

These cases come to us from the States of Kansas, South Carolina,
Virginia, and Delaware…
```

`\readingnote` {⟨*note-text*⟩}

Adds a footnote to the reading's heading. This command *must come before* the `\reading` command.

`\reading` [⟨*short-name*⟩] {⟨*name*⟩}

Creates a section heading starting a reading. The title of the reading is given as ⟨*name*⟩, and a short Table of Contents version may be given as ⟨*short-name*⟩.

As a convenience, if ⟨*name*⟩ starts with *In re* or contains *v.*, the name (and short name) will automatically be italicized for being a case name.

Sectioning and Table of Contents format for readings.

`\readingcite` {⟨*citation*⟩}

Produces a second heading line below a `\reading` entry, that gives the citation for the reading text.

`\opinion` {⟨*text*⟩} `\par`

Formats the line where the opinion author is given. The argument need not be in braces; it is terminated at the end of the paragraph.

`\readinghead` {⟨*text*⟩}

Creates a heading inside a reading.

4.2 Statute and Question Environments

`statute` Formats text for an indented statute's subsections. Statutes are typically formatted as indented paragraphs, with higher levels of indentation pushing the right margin but retaining the indentation structure. (Statutes are typically not formatted with hanging indentation.)

This environment provides for such indentation, for the second and higher levels. (The first level is simply normal paragraph indentation and thus requires no environment.) Each paragraph should be preceded by an `\item` command.

(This environment is currently not very well tested and ought to be improved.)

`questions` [⟨*title*⟩]

Creates an environment for notes and questions. The title of the environment is by default "Notes and Questions," and may be changed with the optional argument. If the optional argument is empty, no heading is produced.

The contents of the environment should be a list with `\item` commands.

4.3 Fonts

Two fonts are used throughout the casebook, one for editorial materials and one for readings. The following rules are used to distinguish the two:

- Files starting with `intro-` or `narrative-`, or files ending with `-qs`, are editorial material; anything else is a reading.

- Block quotes are always assumed to be readings.
- Footnotes follow their own rules, described below.

`\edfont` `\readingfont` The fonts may also be manually selected, with the commands `\edfont` (for editorial material) and `\readingfont` (for readings). Note that the `\readings` command *does not apply the reading font.* This is because some editors like to include Notes or other materials with a reading-like heading. Such material should be included in an editorially-named file (`narrative-`⟨*file*⟩`.tex`), and it will be set in the editorial font.

`\EditorialFont` `{`⟨*font-commands*⟩`}`

Executes ⟨*font-commands*⟩ for any editorial material. By default, editorial material is set in a sans serif font.

`\ReadingFont` `{`⟨*font-commands*⟩`}`

Executes ⟨*font-commands*⟩ for any reading material. By default, reading material is set in a serif font.

`\ifedmaterial` A conditional for determining whether the current text is a reading or editorial material.

`\HeaderFonts` `{`⟨*pagenum*⟩`} {`⟨*left*⟩`} {`⟨*right*⟩`}`

Sets the fonts for the running header. As described below in the Implementation section, the running header consists of a chapter name on the left and the current reading on the right. By default, sans serif fonts are used, with the chapter set in small caps.

`\EditorMark` `{`⟨*note-text*⟩`}`

Transforms ⟨*note-text*⟩ with an indication that the text originated from an editor. By default, this just appends "—Eds.", and the macro can be redefined as desired.

`\footnote` `{`⟨*note-text*⟩`}`

Regular footnotes can only be used in editorial material, and are editorial material themselves.

4.4 Graphics

The following commands are provided for inclusion of graphics. Graphics files may be located either in a module directory or in a separate `images` directory in a repository. The extension `.png`, `.jpg`, or `.pdf` may be omitted from the filename.

`\usegraphic` `[`⟨*options*⟩`]` `{`⟨*filename*⟩`}`

Include a graphic in the current position inline with text. The ⟨*options*⟩ are those options available for the `\includegraphics` command of the `graphicx` package.

By default, graphics take up a maximum of 30% of the text height and 80% of the text width. The optional argument to any of the graphics inclusion macros can change that.

`\heregraphic` `[`⟨*options*⟩`]` `{`⟨*filename*⟩`}`

Centers the graphic at the current position in the text.

`\captionedgraphic` `[`⟨*options*⟩`]` `{`⟨*filename*⟩`}` `{`⟨*caption*⟩`}`

Places the graphic in a floating figure with a caption. A cross-reference label of `f:`⟨*filename*⟩ is automatically attached to the figure number.

Because captions are always editorial material (unless specified otherwise), they are displayed in the editorial font.

`\GraphicsDirectory` `{`⟨*directory*⟩`}`

Specifies the directory within repositories where images may be found. By default, it is `images`.

5 Implementation

The remaining text describes internal operations of the package, and need not be read unless the package is doing something unexpected and you want to fix it.

5.1 Finding Module Files

`\mbk@import@newmod` `{`⟨*module*⟩`}` `{`⟨*filename*⟩`}`

Imports where a module name is given.

`\mbk@import@nomod` {⟨*filename*⟩}

Imports where no module name is given.

`\mbk@try@modfile` {⟨*module*⟩} {⟨*filename*⟩}

Tries to find a module file, across all the repositories.

5.2 Tracking the Current Module and File

Module imports are tracked via a stack, so it is always possible to know which module is in current use. (The normal TeX grouping mechanism cannot be used, because otherwise content would be included inside groups.)

`\mbk@module@cur` The current module.

`\mbk@current@file` The current file.

`\mbk@module@stack` The stack of module inclusions. The list is comma-separated and always ends in a comma.

`\mbk@module@push` {⟨*module*⟩}

Push a module onto the stack.

`\mbk@module@pop` Delete a module from the stack. If one tries to pop the last module from the stack, this macro will generate an argument error (there won't be enough commas).

Implementation: File Inclusion

`\mbk@try@file` {⟨*filename*⟩} {⟨*content*⟩}

Tries including a file among several. Several `\mbk@try@file` commands may be included in sequence, terminated with `\mbk@try@file@end`. If the file exists, then {⟨*content*⟩} is inserted and any other material up to `\mbk@try@file@end` will be discarded.

`\mbk@try@file@default` {⟨*content*⟩} `\mbk@try@file@end`

What to do if no `\mbk@try@file` commands succeed. All material up to `\mbk@try@file@end` is used.

5.3 Cross-Reference Checking

`\mbk@register@file` {⟨*module*⟩} {⟨*filename*⟩}

To implement cross-reference checking, every file is "registered" at the time it is imported. The registration confirms any assertions that can be determined upon registration, and records information for further checking.

`\mbk@register@pre` {⟨*filename*⟩}

Marks that a file will be included at a later time.

5.4 Fonts

`\@defaultfamilyhook` This code hooks into the LaTeX command that resets the default font, forcing editorial or reading font selection every time the font is reset.

`\mbk@formatting@for` {⟨*filename*⟩}

Selects the font based on the filename.

`quotation`
`quote` Redefine the `quote` and `quotation` environments to use the reading font.

5.5 Document-Level Structure

The introduction of readings as a document section type requires some modifications to the usual LaTeX document structure.

First, the package creates a new running head format, where the chapter name is placed on the left and the current reading is placed on the right.

Readings are section level 4, and paragraphs/subparagraphs are placed below that level. Numbering continues up through level 3 (i.e., readings are not numbered), and the Table of Contents includes readings.