Producing HTML directly from LaTeX: the lwarp package

Brian Dunn
bd@BDTechConcepts.com

March 7, 2017

Abstract

The lwarp package allows LaTeX to directly produce HTML5 output, using external utility programs only for the final conversion of text and images. Math may be represented by SVG files or MathJax.

Documents may be produced by LaTeX, LuaLaTeX, or XeLaTeX. A `texlua` script removes the need for system utilities such as `make` and `gawk`, and also supports `xindy` and `latexmk`. Configuration is automatic at the first manual compile.

Print and HTML versions of each document may coexist, each with its own set of auxiliary files. Support files are self-generated on request.

A modular package-loading system uses the lwarp version of a package for HTML when available. Several dozen LaTeX packages are supported with these high-level source compatibility replacements.

A tutorial is provided to quickly introduce the user to the major components of the package.

Why LaTeX

Before attempting to justify yet another LaTeX-to-HTML conversion package, it may be worth stepping back for a moment to consider LaTeX itself. A quick web search for "LaTeX vs Word", or some other program, will return many web pages and discussion threads comparing the various programs and their advantages. Things change, however, and many of these discussions are now obsolete due to modern advances in each program's capabilities. As examples, LaTeX no longer has many problems dealing with fonts, and LyX plus a number of integrated development environments are now available, along with online collaborative-development websites [1, 2, 3, 4]. Meanwhile, Microsoft® Word can typeset nice mathematics with a LaTeX-ish input and has improved in its typesetting and stability, and commercial page-layout programs have improved in their handling of large documents.

Nevertheless, many of the traditional advantages of LaTeX still apply: the visibility, stability, and portability of plain-text markup, regular-expression search and replace of both text and formatting commands, easy revision control, the ability to handle large and complex documents, extensive programming capabilities, and the large number of user-supplied packages solving real-world problems. In many cases, it's still faster to type a few arguments than it is to open a dialog box and select and fill in entries, and a powerful programming text editor is usually more responsive than a word processor.

Another development is the large number of markup languages now available, usually with a number of options for output format. These systems are based on plain-text markup using inline tags or sequences of special characters, and thus share some of the advantages of LaTeX. While these systems are useful for smaller documents, cross-referencing is limited (although the AsciiDoc syntax does offer full cross-referencing to figures and tables), much of the customization is done at the back end, and the syntax of special symbols tends to become rather dense once things become complicated. LaTeX has the advantage of giving macros relatively readable names.

Great progress has been made in making LaTeX more widely accessible. Online collaborative LaTeX editing websites now claim a million users and thousands of institutions, and LaTeX is also now available as a browser application [5]. If anything, LaTeX seems to be building momentum, even after all these decades.

Why convert LaTeX to HTML

Unfortunately, modern publishing often involves submission and rounds of editing in Word format, conversion to an XML intermediate, then conversion yet again to a professional typesetting system, along with HTML or EPUB versions. Each of these stages may be performed by different groups of people in different parts of the world [6], most of whom are not familiar with the technical content, and also by imperfect algorithms whose programmers haven't thought of every possibility. (Example: An incorrect line break in a superscript, where a hyphen had been used as a minus sign.) The resulting errors are often beyond the author's control — the final product having problems which were not present in the signed-off proof.

While it is unrealistic to expect any of this to change, there is a movement towards self-publishing [7, 8, 9, 10] which removes many of these problems while also providing the benefits of quick turnaround, print on demand, and the ability to make changes or updates as needed. This requires the ability to create a professional-quality printed document in several sizes (e-tablet included), which LaTeX certainly can provide, but also the ability to create HTML or EPUB as well. Providing a high-quality PDF version is better than asking the user to print from HTML, whereas providing an HTML version provides easy accessibility and some search-engine benefits. Providing both is the best option.

Another application of LaTeX-to-HTML conversion is the creation of an informational (non-interactive) website. Many scientists, professors, and engineers would benefit from having their own website on which their own technical papers could be published, and they could apply their pre-existing LaTeX skills not just to the documents but also to the website itself.

Why LaTeX is hard to convert

Modern HTML5 and CSS3 are quite capable, to the point where they can be used to produce technical books [11]. Nevertheless, there are some practical problems to overcome in order to create a good conversion from LaTeX to HTML.

One of the first issues is the difference between individual printed pages versus the HTML concept of an endless scroll of variable width. Footnotes can become endnotes, but `\pageref` refers to *what*, exactly? Is `\linewidth` for the current screen size, or is it for a conceptual page size? The relationship between font size, image size, and screen size is broken, there is no margin for marginpars, and text may be reflowed at any time.

LaTeX knows about stretchable space, which is not true of HTML. A `\vfill` is almost meaningless in HTML, and an `\hfill` is not much better. Nor do floating objects translate well, since there are no page breaks at which to place them.

Math in HTML has been a problem for years, and the MathML standard has not been adopted by many browsers [12, 13, 14]. MathJax is nice and getting better all the time, but requires JavaScript and web access or a local copy, possibly making it unacceptable for use in EPUB documents [15], and it can be relatively slow. Drawing math as images has its own limitations.

Aside from display-related issues, another general problem with converting LaTeX to HTML is the fact that LaTeX does not use end delimiters for many of its syntactic units. A `\section` does not have an `\endsection` before the next `\section`, for example, and beginning the next `\section` may first require closing several nested levels worth of currently open subsections and paragraphs. Nor does `\bfseries` have a syntactically defined endpoint, and HTML/CSS do not support state switching.

Finally, LaTeX engines do not allow for the direct plain-text output of HTML tags and text content, thus requiring some kind of PDF-to-text conversion, followed by some system to optionally split the results into separate web pages of HTML, and also copy out any inline images which must be cropped and converted for web display.

Existing methods

Several methods already exist for converting some subset of LaTeX into HTML. These are discussed in slightly more detail in the lwarp manual.

The closest to lwarp in design principle is the internet class by Andrew Stacey [16], an interesting project which directly produces several versions of markdown, and also HTML and EPUB.

There is also the TeX4ht project [17], which uses LaTeX itself to do most of the work, along with an external program to convert special codes into HTML or several other formats.

A number of other projects use an intermediate translation program to parse LaTeX source and then convert it externally. See HEVEA [18], TTH [19], GELLMU [20], LaTeXML [21], plasTeX [22], LaTeX2HTML [23], and TeX2page [24], most of which are found on CTAN.

GladTeX [25] may be used to insert LaTeX math expressions into pre-existing HTML code.

For sake of completeness, it should be mentioned that there are plugins allowing the entry of LaTeX math expressions for Word [26, 27] and LibreOffice™ [28], as well as commercial page-layout programs.

Why another approach

Nothing except LaTeX truly understands LaTeX.

More to the point, it's easier for LaTeX to program HTML than for a third-party converter program to understand LaTeX. A larger portion of LaTeX and its associated packages can be parsed and converted when LaTeX itself does the work. Another advantage of staying with LaTeX alone is that development of the core and additional packages can be done without requiring skills in an additional language.

Development

AsciiDoc markup

The initial inspiration for the lwarp package was the internet class. Seeing that someone else had trained LaTeX to produce markup, it was decided to program LaTeX to generate the AsciiDoc markup syntax. AsciiDoc has several advantages over other markup languages, including improved cross-references, and its Asciidoctor variant generating modern HTML5 output. Using AsciiDoc as an intermediate syntax lifted much of the conversion load from LaTeX, while providing almost all of the functionality which would be required for a typical technical paper. Nevertheless, AsciiDoc just couldn't represent many of the concepts commonly-used in LaTeX. Tabular material and minipages were limited, and the toolchain was a bit of a chore to handle. Thus, the need to program LaTeX to directly produce HTML.

Low-level and high-level patches

In most cases, code is patched at the lowest level possible, allowing for increased code compatibility and reuse. The process of finding the best place to patch code resulted in several waves of revisions, especially in the areas of floats, auxiliary files, and package handling.

Entire packages must be supported. User-level macros, counters, and so on are intercepted and redirected or ignored as necessary.

Fonts and encodings

A vector-based font must be used for `pdftotext` to convert the PDF to plain text. A roman face is used in most cases, which preserves em-dashes with `pdflatex`. The HTML tags are printed to the PDF file in a monospaced font, and the quote marks must be upright quotes, but this breaks the em-dash in `pdflatex`.

LaTeX can display many specialized glyphs which are not encoded and thus won't be picked up by `pdftotext`. It may be possible to assign these using `glyphtounicode.tex` or newunicodechar. For many uses `lualatex` or `xelatex` will be preferred, as `pdftotext` can use UTF-8 encoding.

The chosen font will be visible in HTML when rendering math as SVG images.

Page layout

While generating HTML, a very small font is used and the page layout is changed to allow generous margins. Both are to avoid overflow, which can become a problem with long HTML expressions. Ragged right is used to avoid hyphenation. The `\linewidth` is set for a virtual six-inch wide document, which solves problems where the user specifies a fraction of `\linewidth` for graphics images or tabular columns.

Paragraph handling

Each paragraph in HTML must be enclosed in an opening and closing tag. To track paragraphs, the `\everypar` hook triggers an action when a paragraph starts, and `\par` is re-assigned to close paragraphs. Flags are used to control whether to turn tag creation on or off in certain circumstances. For example, inside an HTML `<span>` paragraph tags are not allowed, but a `<br>` tag may still be used for something like a multiline caption.

Sectioning

HTML sectioning requires nesting and unnesting LaTeX sectional units. Since there are no section-ending LaTeX commands, each `\chapter`, `\section`, etc.

must first un-nest any previously nested sectional units up to its own level. A simple LIFO stack is used to track section depths and closing tags.

The sectioning code is one area which was rewritten for HTML output, rather than try to reuse something which is patched by so many packages. Section breaks may trigger a new HTML file, and automatic cross-referencing occurs as well. Formatting and paragraph handling depend on which kind of section it is.

Cross-referencing

While the LaTeX and cleveref cross-referencing code is used, additional referencing is required to track HTML pages and `id` tags. Automatically-generated tags are used for each section and float, allowing cross-references to link to specific objects on each page. Indexing uses the `xindy` program to generate HTML tags.

Floats

The combination of caption, subcaption, and newfloat packages is supported. These were chosen from among the many alternatives due to being commonly used, flexible, and kept up-to-date. Floats are generated in place, as if they were declared `[H]`ere. Support is provided for other packages, such as float, floatrow, capt-of, wrapfig, placeins, and the author's own keyfloat which can also support margin floats.

Image generation

Math, `picture` environments, Ti*k*Z, and anything else with graphic content may be placed inside a special `lateximage` environment. When this environment is started, an HTML open comment tag is generated, followed by a new page. The contents of the graphic environment are then drawn on the empty page, followed by yet another new page whose first line is an HTML closing comment tag. The comment tags encapsulate any text contents of the graphics page such that they are not displayed in the HTML page. Meanwhile, the page and image numbers are written to a text file to be processed by `lwarpmk`, which later separates the PDF file into individual graphics files, each of which is then cropped, converted to SVG, and named, ready for inclusion in the final web page. Finally, HTML instructions are generated to load the resulting graphics file at that position in the web page. Paragraph and formatting elements must be restored to their LaTeX meaning during the creation of the graphic.

Math

Math may be represented by SVG images using the `lateximage` environment, with the LaTeX source embedded as HTML `alt` tags, or by using the MathJax script.

Graphics images

Graphics images may be included at a specified width and/or height, or as a fraction of `\linewidth`. When `\linewidth` is used, the assumed six-inch line is used as well, and the final image size is fixed in HTML, along with a `max-width` CSS property to hopefully avoid requiring the user of a hand-held device to pan across the image.

graphicx is emulated quite well, although the HTML standard does not agree with LaTeX about white space while rotating or scaling, so expect ugly results when doing so.

Minipages

Minipages are created using `inline-flex`, a fairly new CSS3 property which allows side-by-side minipages with a vertical alignment. Unfortunately, a minipage inline with a paragraph of text cannot work since HTML does not allow a block inside a paragraph, so the minipage then goes onto its own line. Furthermore, a `<div>` cannot be used inside a `<span>`, so lwarp disables minipages inside spans, although `\newline` or `\par` can be used to create a `<br>` tag.

For those cases where the user may wish to have an HTML minipage without a fixed width, the new command `\minipagefullwidth` declares that the following minipage may be the natural width of its contents, up to the full width of the display. During print output, the minipage will still use its assigned width.

Tabular

Tabular material is a challenge, no matter the syntax. This is one area where lwarp had to totally replace the original code rather than try to patch the existing. Data arrays in the computer-science sense had to be used to track column types, as well as actions for `\>`, `\<`, and `\@`. Border-generation logic had to be created. As of this writing vertical rules are not supported, but booktabs are, except for trim options which would be very hard to do in CSS.

Navigation

In an attempt to avoid resorting to JavaScript, a "sideTOC" has been developed. This is a subset of the table of contents which appears at the side or top of each web page. At present this sideTOC is not in its own pane, which has both its advantages and disadvantages, and this may be changed in the future. To provide for "responsive web design", the sideTOC is moved to the top of the page when the display is narrow, and an additional Home button is placed at the bottom as well.

Package handling

A major design decision was made regarding handling the loading of additional packages. Some packages may be used as-is, some must be ignored, and some must be patched in some way to be usable for HTML. Furthermore, it would be best if these actions were separated from the lwarp core, interacted well with each other, and expandable by the user.

To provide for all of this, lwarp intercepts both the `\usepackage` and `\RequirePackage` macros to first see if there is an lwarp-provided alternative package. If so, that version is used instead of the original. It is up to the lwarp version of the package to either totally ignore the original, or load the original with its options and then perform additional patches or other actions afterward.

Several dozen packages are already supported by lwarp, including some of the most commonly used in all major categories. For packages which lwarp does not yet handle, the user may apply the print-only environment or macro to encapsulate things which do not apply to HTML. The user may also wish to create a custom package for lwarp to use, containing nullified macros and environments, along with any booleans, counters, and lengths which may be used in the source code. Such a package should be named

`lwarp-packagename.sty`

and then lwarp will use it whenever the document calls for `packagename.sty` while creating HTML.

Using lwarp

The following is an overview of the configuration and use of lwarp. Major advances have been made in simplifying this process, including the above-mentioned package handling code. As a result, the user may simply add the lwarp-newproject and lwarp packages to the code at the correct place, compile the document once in the traditional way, and then immediately use the `lwarpmk` utility for further print or HTML versions.

Project setup — lwarp-newproject

Previous versions of lwarp required the user to copy or link a number of configuration files and scripts, and also modify a `makefile`.

Recent improvements include the use of automatic detection of the TeX engine, operating system, and jobname. These are written to a general configuration file for the new `lwarpmk` program. `lwarpmk` is a utility used to compile print and HTML versions of the document.

Furthermore, the lwarp-newproject package is provided, to be loaded just before lwarp. This package writes various additional configuration and utility files. Included are a project-specific configuration file for the lwarpmk utility (thus allowing multiple documents to reside in the same folder), a configuration file for xindy, a number of .css files, and a fragment of JavaScript used to invoke MathJax.

Also written is a new ⟨*project*⟩_html.tex file, whose name is the project's \jobname with _html appended. This is a small file which simply sets a few options to select HTML conversion, then \input s the user's document. In this way, a compile of the user's document generates a print version, while a compile of the _html version generates an HTML version. Both versions and their auxiliary files coexist. The lwarp-newproject package is only active when compiling the print version, and the configuration files are regenerated each time the print version is recompiled. Should the user wish to switch TEX engines, the approach is to remove the auxiliary files, then manually recompile the main document using pdflatex, lualatex, or xelatex. This engine will then be used by the lwarpmk utility for future compiles of either the print or HTML version.

The lwarpmk utility program is to be provided as a LuaTEX executable by the TEX distribution, but it is possible that someone may wish to archive it along with the project. For this purpose, an option for the lwarp-newproject package is available to cause a write of a local copy of lwarpmk.

The CSS files include a master lwarp.css file which provides the essential functions and a basic LATEX-ish style, along with optional CSS files for a more formal or a more contemporary style. Also created is sample-project.css, which shows how to load one of the provided CSS files and also provides a place to make modifications. This file is to be renamed, as it will be overwritten by lwarp-newproject each time a print version is created.

Compiling the document — lwarpmk

Previous versions of lwarp relied on the make, gawk, and grep utilities. Fortunately, modern TEX distributions provide the LuaTEX program — an extension of the Lua programming language. The use of LuaTEX to provide the required utility functions eases issues of availability, installation, and portability.

lwarpmk's configuration file tells it the operating system, the TEX engine, the source \jobname, the filename of the homepage, and whether the latexmk utility should be used to compile, or whether lwarpmk should detect changes and recompile by itself.

lwarpmk is able to compile the printed or HTML version of the document, process the index for the printed or HTML version, request a recompile, process the lateximage files, clean the auxiliary files, or process the PDF into HTML files (a subset of its functionality, intended to be used by a makefile if desired).

If a document name is provided, lwarpmk processes that document according to its project-specific configuration file, otherwise it uses its general configuration file to reprocess the last document.

Several utility programs are still required for the HTML conversion. `pdftotext` is used to convert the PDF document into UTF-8 text. `pdfseparate` extracts individual graphic images from the PDF file, `pdfcrop` crops these images, and `pdftocairo` is used to convert PDF images into SVG images. `pdfcrop` is provided as part of the TEX distribution, and the rest are commonly-available utilities from the Poppler project, and should be made available by the operating system's package manager.

Customizing the HTML

Aside from the CSS files, additional customization is provided by a number of user-adjustable settings and macros.

HTML files may be numbered or named, and a prefix may be applied to each file. The homepage may have its own name. Counters control the depth of the sideTOC and the file division.

Files may be split by the strict sectioning depth level, or higher levels may be combined into one file. For example, a part, its first chapter, and its first section may be combined into one file while further files are split at the section level until the next part or chapter.

The HTML `lang` attribute may be set for the document. The CSS file and HTML `description` may be changed at each file split.

Programming hooks are provided for the top of the home page, the top of other pages, and the bottom of all pages. These are useful for logos, copyright notices, and contact information.

Special environments and macros are provided for functions which should be applied to only the printed or only the HTML versions of the document.

Tutorial

A tutorial is provided which quickly guides the user through the setup of a document, compiling printed and HTML versions, processing graphics images, generating math in SVG or MathJax format, customizing the HTML, using `latexmk`, switching the TEX engine, processing multiple documents in the same directory, and cleaning the auxiliary files.

References

[1] ShareLaTeX. https://www.sharelatex.com

[2] Overleaf. https://www.overleaf.com

[3] Authorea. https://www.authorea.com

[4] Papeeria. https://papeeria.com/

[5] LaTeX Base. https://latexbase.com

[6] Forum topic. ResearchGate, “Why is LaTeX not used as an end-to-end solution in the publishing industry?”, 2013, https://www.researchgate.net/post/Why_is_LaTeX_not_used_as_an_end-to-end_solution_in_the_publishing_industry

[7] Open Education Database, “The Academic’s Guide to Self-Publishing”, 2014, http://oedb.org/ilibrarian/the-academics-guide-to-self-publishing/

[8] Dennis Meredith, Research Explainer, “Self-Publishing Series”, 2013, https://researchexplainer.com/2013/06/26/self-publishing-series-i-making-the-decision/

[9] Robert Ghrist, “Why I published my mathematics text print-on-demand via Amazon’s Createspace”, 2014, https://www.math.upenn.edu/~ghrist/whyselfpublish.html

[10] Nicola L. C. Talbot, Dickimaw Books, “Self-Publishing”, 2014, http://www.dickimaw-books.com/nonfiction/self-publishing/

[11] Sanders Kleinfeld, “Next-Generation Book Publishing: Of the HTML, by the HTML, for the HTML”, Digital Book World, 2014, http://www.digitalbookworld.com/2014/next-generation-book-publishing-of-the-html-by-the-html-for-the-html/

[12] Christian Lawson-Perfect, “Dark days for MathML support in browsers”, The Aperiodical, 2013, http://aperiodical.com/2013/11/dark-days-for-mathml-support-in-browsers/

[13] Stephen Shankland, “Google subtracts MathML from Chrome, and anger multiplies”, CNET, 2013,

https://www.cnet.com/news/google-subtracts-mathml-from-chrome-and-anger-multiplies/

[14] Neil Soiffer, "Microsoft cripples the display of math in IE10 & 11", Design Science News, 2013, http://news.dessci.com/microsoft-cripples-display-math-ie10-11

[15] "EPUB3 Reading systems overview", The MathJax Consortium, 2015, http://docs.mathjax.org/en/latest/misc/epub.html

[16] Andrew Stacey, "latex-to-internet", https://github.com/loopspace/latex-to-internet

[17] TEX4ht: LaTeX and TeX for Hypertext, http://tug.org/tex4ht

[18] HEVEA, http://hevea.inria.fr/

[19] TTH, http://hutchinson.belmont.ma.us/tth/

[20] William F. Hammond, "GELLMU — Introductory Survey", http://www.albany.edu/~hammond/gellmu/

[21] LaTeXML — A LaTeX to XML/HTML/MathML Converter, http://dlmf.nist.gov/LaTeXML/

[22] plasTeX, http://tiarno.github.io/plastex/

[23] LaTeX2HTML, http://www.latex2html.org/

[24] Dorai Sitaram, TeX2page, https://ds26gte.github.io/tex2page/

[25] Martin Gulbrandsen and Sebastian Humenda, GladTeX, https://humenda.github.io/GladTeX/

[26] "LaTeX in Word", https://github.com/EngineeroLabs/latex_in_word

[27] TeXsword, https://sourceforge.net/projects/texsword/

[28] Roland Baudin, "TexMaths, a LaTeX Equation Editor for LibreOffice", http://roland65.free.fr/texmaths/

Trademarks

LibreOffice is a trademark of The Document Foundation.

MathJax is copyright 2009 and later. The MathJax Consortium is a joint venture of the American Mathematical Society (AMS) and the Society for Industrial and Applied Mathematics (SIAM) to advance mathematical and scientific content on the web.

Microsoft® is a registered trademarks of Microsoft Corporation in the United States and/or other countries.

TeX is a trademark of American Mathematical Society.