The make4ht build system

Michal Hoftich*

Version 0.2a
2018-05-03

Contents

*michal.h21@gmail.com

1 Introduction

`make4ht` is a simple build system for `tex4ht`, TEX to XML converter. It provides a command line tool that drive the conversion process. It also provides a library which can be used to create customized conversion tools. An example of such conversion tool is tex4ebook for conversion of TEX to ePub and other e-book formats.

1.1 How it works

1.1.1 The issues with default `tex4ht` conversion commands

`tex4ht` system supports several output formats, most notably `XHTML`, `HTML 5` and `ODT`. The conversion can be invoked using several commands. These commands invoke LaTeX or Plain TeX with special instructions to load `tex4ht.sty` package. The TEX run produces special `DVI` file which contains the code for desired output format. The `DVI` file is then processed and desired output files are created.

The basic command provided by `tex4ht` is named `htlatex`. It compiles LATEX files to `HTML` with this command sequence:

```
latex $latex_options 'code for loading tex4ht.sty \input{filename}'
latex $latex_options 'code for loading tex4ht.sty \input{filename}'
latex $latex_options 'code for loading tex4ht.sty \input{filename}'
tex4ht $tex4ht_options filename
t4ht $t4ht_options filename
```

The options for various parts of the system can be passed on the command line:

```
htlatex filename "tex4ht.sty options" "tex4ht_options" "t4ht_options" "latex_options"
```

For basic `HTML` conversion it is possible to use the most basic invocation:

```
htlatex filename.tex
```

It can be much more involved for HTML 5 output in UTF-8 encoding:

```
htlatex filename.tex "xhtml,html5,charset=utf-8" "-cmozhtf -utf8"
```

make4ht can simplify it:

```
make4ht -uf html5 filename.tex
```

Another issue is the fixed compilation order and hard-coded number of LaTeX invocations.

When you need to run a program which interact with LaTeX, such as Makeindex or Bibtex, you need to create a new script based on htlatex, or run htlatex twice, which means that LaTeX will be invoked six times. This can lead to significantly long compilation times. make4ht provides build files and extensions, which can be used for interaction with external tools.

It is also possible to have several compilation modes. When you just add new text to a document, which doesn't contain cross-references, don't add new stuff to the table of contents, etc., it is possible to use the draft mode which will invoke LaTeX only once. It can save quite a lot of the compilation time:

```
make4ht -um draft -f html5 filename.tex
```

There are also issues with a behaviour of the t4ht application. It reads file filename.lg, generated by tex4ht, where are instructions about generated files, CSS instructions, calls to external applications, instructions for image conversions etc. It can be instructed to copy generated files to some output directory, but it doesn't preserve directory structure, so when you have images in a subdirectory, they will be copied to the output directory. Links will be pointing to a non-existing subdirectory. The following command should copy all output files to the correct destinations.

```
make4ht -d outputdir filename.tex
```

The image conversion is configured in the env file, which has really strange syntax based and the rules are os dependent. make4ht provides simpler means for the image conversion in the build files.

With make4ht build files, we have simple mean to fix these issues. We can change image conversion parameters without the need to modify the env file, or execute actions on the output files. These actions can be either external programs such as xslt processors or HTML tidy or Lua functions.

The idea is to make system controlled by a build file. Because Lua interpreter is included in modern TeX distributions and Lua is ideal language for such task, it was chosen as language in which the build scripts are written.

2 Output file formats and extensions

The default output format used by make4ht is html5. Different format can be requested using the --format option. Supported formats are:

- `xhtml`
- `html5`
- `odt`

The `--format` option can be also used for extension loading.

2.1 Extensions

Extensions can be used to modify the build process without the need to use a build file. They may post-process the output files or request additional commands for the compilation.

The extensions can be enabled or disabled by appending `+EXTENSION` or `-EXTENSION` after the output format name:

```
make4ht -uf html5+tidy filename.tex
```

Available extensions:

latexmk_build use `Latexmk` for LaTeX compilation.

tidy clean the `HTML` files using the `tidy` command.

common_filters clean the output HTML files using filters.

common_domfilters Clean the HTML file using DOM filters. It is more powerful than `common_filters`. Used DOM filters are `fixinlines`, `idcolons` and `joincharacters`.

mathjaxnode use mathjax-node-page to convert from MathML code to HTML + CSS or SVG. See the available settings.

staticsite build the document in form suitable for static site generators like Jekyll.

3 Build files

`make4ht` supports build files. These are `Lua` scripts which can adjust the build process. You can request external applications like `bibtex` or `makeindex`, pass options to the commands, modify the image conversion process, or post-process the generated files.

`make4ht` tries to load default build file named as `filename + .mk4 extension`. You can choose different build file with `-e` or `--build-file` command line option.

Sample build file:

```
Make:htlatex()
Make:match("html$", "tidy -m -xml -utf8 -q -i ${filename}")
```

`Make:htlatex()` is preconfigured command for calling LaTeX with `tex4ht` loaded on the input file. In this case, it will be called one time. After compilation, the `tidy` command is executed on the output `HTML` file.

Note that you don't have to call `tex4ht` and `t4ht` commands explicitly in the build file, they are called automatically.

3.1 User commands

You can add more commands like `Make:htlatex` using `Make:add` command:

```
Make:add("name", "command", {parameters}, repetition)
```

The `name` and `command` parameters are required, rest of the parameters are optional.

This defines `name` command, which can be then executed as `Make:name()` command.

3.1.1 Provided commands

`Make:htlatex` One call to TeX engine with special configuration for `tex4ht` loading.

`Make:latexmk` Use `Latexmk` for the document compilation. `tex4ht` will be loaded automatically.

`Make:tex4ht` Process the `DVI` file and creates the output files.

`Make:t4ht` Creates the CSS file.

3.1.2 Command function

The `command` parameter can be either string template or function:

```
Make:add("text", "echo hello, input file: ${input}")
Make:add("function", function(params)
  for k, v in pairs(params) do
    print(k..": "..v)
  end, {custom="Hello world"}
)
```

The template can get variable value from the parameters table using a `${var_name}` placeholder. Templates are executed using operating system, so they should invoke existing OS commands. Function commands may execute system commands using `os.execute` function.

3.1.3 Parameters table

`parameters` parameter is optional, it can be table or `nil` value, which should be used if you want to use the `repetition` parameter, but don't want to modify the parameters table.

The table with default parameters is passed to all commands, they can be accessed from command functions or templates. When you specify your own parameters in the command definition, these additional parameters are added to the default parameters table for this particular command. You can override the default parameters in the parameters table.

The default parameters are following:

`htlatex` used compiler

`input` it is output file name in fact

`tex_file` input TeX file

`latex_par` parameters to `latex`

`packages` insert additional LaTeX code which is inserted before `\documentclass`. Useful for passing options to packages or additional packages loading

`tex4ht_sty_par` parameters to `tex4ht.sty`

`tex4ht_par` parameters to the `tex4ht` application

`t4ht_par` parameters to the `t4ht` application

`outdir` the output directory

`repetition` limit number of command execution.

`correct_exit` expected `exit code` from the command. The compilation will be terminated if the command `exit code` is different.

3.1.4 Repetition

Repetition is number which specifies a maximal number of executions of the particular command. This is used for instance for `tex4ht` and `t4ht` commands, as they should be executed only once in the compilation. They would be executed multiple times if you include them in the build file because they are called by `make4ht` by default. Because these commands allow only one `repetition`, the second execution will be blocked.

3.1.5 Expected exit code

You can set the expected exit code from a command with a `correct_exit` key in the parameters table. The compilation will be stopped when the command returns a different exit code.

This mechanism isn't used for LaTeX (for all TeX engines and formats, in fact), because it doesn't differentiate between fatal and non-fatal errors, and it returns the same exit code in all cases. Log parsing is used because of that, error code `1` is returned in the case of fatal error, `0` is used otherwise. The `Make.testlogfile` function can be used in the build file to detect compilation errors in the TeX log file.

3.2 File matches

Another type of action which can be specified in the build file is `match`. It can be called on the generated files:

```
Make:match("html$", "tidy -m -xml -utf8 -q -i ${filename}")
```

It tests output file names with `Lua` pattern matching and on matched items will execute a command or a function, specified in the second argument. Commands may be specified as strings, the templates will be expanded, `${var_name}` placeholders will be replaced with corresponding variables from the `parameters` table, described in the previous subsection. One additional variable is available: `filename`. It contains the name of the current output file.

The above example will clean all output `HTML` files using the `tidy` command.

If function is used instead, it will get two parameters. The first one is a current filename, the second one table with parameters.

3.2.1 Filters

Some default `match` actions which can be used are available from the `make4ht-filter` module. It contains some functions which are useful for fixing some `tex4ht` bugs or shortcomings.

Example:

```
local filter = require "make4ht-filter"
local process = filter{"cleanspan", "fixligatures", "hruletohr"}
Make:htlatex()
Make:htlatex()
Make:match("html$",process)
```

The `make4ht-filter` module return a function which can be used for the filter chain building. Multiple filters can be chained, each of them can modify the string which was modified by the previous filters. The changes are then saved to the processed file.

Built-in filters are:

cleanspan clean spurious span elements when accented characters are used

cleanspan-nat alternative clean span filter, provided by Nat Kuhn

fixligatures decompose ligatures to base characters

hruletohr `\hrule` commands are translated to series of underscore characters by `tex4ht`, this filter translate these underscores to `<hr>` elements

entites convert prohibited named entities to numeric entities (currently, only ` `, as it causes validation errors, and `tex4ht` is producing it sometimes)

fix-links replace colons in local links and `id` attributes with underscores. Some cross-reference commands may produce colons in internal links, which results in validation error.

mathjaxnode use mathjax-node-page to convert from MathML code to HTML + CSS or SVG. See the available settings.

staticsite create HTML file in format suitable for static site generators such as Jekyll

svg-height some SVG images produced by `dvisvgm` seem to have wrong dimensions. This filter tries to set the correct image size.

Function `filter` accepts also function arguments, in this case this function takes file contents as a parameter and modified contents are returned.

Example:

```
local filter  = require "make4ht-filter"
local changea = function(s) return s:gsub("a","z") end
local process = filter{"cleanspan", "fixligatures", changea}
Make:htlatex()
Make:htlatex()
Make:match("html$",process)
```

In this example, spurious span elements are joined, ligatures are decomposed, and then all letters ‘a’ are replaced with ‘z’ letters.

3.2.2 DOM filters

DOM filters use the `LuaXML` library to modify directly the XML object. This enables more powerful operations than the regex based filters from the previous section.

Example:

```
local domfilter = require "make4ht-domfilter"
local process = domfilter {"joincharacters"}
Make:match("html$", process)
```

Available DOM filters:

aeneas Aeneas is a tool for automagical synchronization of text and audio. This filter modifies the HTML code to support the synchronization.

fixinlines put all inline elements which are direct children of the `<body>` elements to a paragraph.

idcolons replace the colon (`:`) character in internal links and `id` attributes. They cause validation issues.

joincharacters join consecutive `<span>` or `<mn>` elements.

3.2.3 make4ht-aeneas-config package

Companion for the `aeneas` DOM filter is the `make4ht-aeneas-config` plugin. It can be used to write Aeneas configuration file or execute Aeneas on the generated HTML files.

Available functions:

write_job(parameters) write Aenas job configuration to `config.xml` file. See the Aeneas documentation for more information about jobs.

execute(parameters) execute Aeneas.

process_files(parameters) process the audio and generated subtitle files.

By default, the `smil` file is created. It is assumed that there is audio file in `mp3` format named as the TeX file. It is possible to use different formats and file names using mapping.

The configuration options can be passed directly to the functions or set using `filter_settings "aeneas-config" {parameters}` function.

Available parameters:

lang document language. It is interfered from the HTML file, so it is not necessary to set it.

map mapping between HTML, audio and subtitle files. More info bellow.

text_type type of the input. The `aeneas` DOM filter produces `unparsed` text type.

id_sort sorting of id attributes. Default value is `numeric`.

id_regex regular expression to parse the id attributes.

sub_format generated subtitle format. Default `smil`.

Additional parameters for the job configuration file:

- description

- prefix
- config_name
- keep_config

It is possible to generate multiple HTML files from the LaTeX source. For example, `tex4ebook` generates separate file for each chapter or section. It is possible to set options for each HTML file, in particular names of the corresponding audio files. This mapping is done using `map` parameter.

Example:

```
filter_settings "aeneas-config" {
  map = {
    ["sampleli1.html"] = {audio_file="sample.mp3"},
    ["sample.html"] = false
  }
}
```

Table keys are the configured file names. It is necessary to insert them as `["filename.html"]`, because of Lua syntax rules.

This example maps audio file `sample.mp3` to a section subpage. The main HTML file, which may contain title and table of contents doesn't have an corresponding audio file.

The filenames of sub files corresponds to chapter numbers, so they are not stable when a new chapter is added. It is possible to request file names interfered from the chapter titles using the `sec-filename` option or `tex4ht`.

Available `map` options:

audio_file the corresponding audio file

sub_file name of the generated subtitle file

The following options are same as their counter-parts from the main parameters table and generally don't need to be set:

- prefix
- file_desc
- file_id
- text_type
- id_sort
- id_prefix
- sub_format

Full example:

```
local domfilter = require "make4ht-domfilter"
local aeneas_config = require "make4ht-aeneas-config"

filter_settings "aeneas-config" {
  map = {
    ["krecekli1.xhtml"] = {audio_file="krecek.mp3"},
    ["krecek.xhtml"] = false
  }
}

local process = domfilter {"aeneas"}
Make:match("html$", process)

if mode == "draft" then
  aeneas_config.process_files {}
else
  aeneas_config.execute {}
end
```

3.3 Image conversion

It is possible to convert parts of LaTeX input to pictures, it is used for example for math or diagrams in `tex4ht`.

These pictures are stored in a special `dvi` file, which can be processed by the `dvi to image` commands.

This conversion is normally configured in the `env file`, which is system dependent and which has a bit unintuitive syntax. This configuration is processed by the `t4ht` application and conversion commands are called for all pictures.

It is possible to disable `t4ht` image processing and configure image conversion in the build file:

```
Make:image("png$",
"dvipng -bg Transparent -T tight -o ${output}  -pp ${page} ${source}")
```

`Make:image` takes two parameters, pattern to match image name and action. Action can be either string template with conversion command, or function which takes a table with parameters as an argument.

There are three parameters:

- `output` - output image file name
- `source` - `dvi` file with the pictures
- `page` - page number of the converted image

3.4 The mode variable

There is global mode variable available in the build file. It contains contents of the --mode command line option. It can be used to run some commands conditionally. For example:

```
 if mode == "draft" then
   Make:htlatex{}
 else
   Make:htlatex{}
   Make:htlatex{}
   Make:htlatex{}
 end
```

In this example (which is the default configuration used by make4ht), LaTeX is called only once when make4ht is called with draft mode:

```
make4ht -m draft filename
```

3.5 The settings table

You may want to access to the parameters also outside commands, file matches and image conversion functions. For example, if you want to convert your file to the OpenDocument Format (ODT), you can use the following settings, based on the oolatex command:

```
settings.tex4ht_sty_par = settings.tex4ht_sty_par "..",ooffice"
settings.tex4ht_par = settings.tex4ht_par .. " ooffice/! -cmozhtf"
settings.t4ht_par = settings.t4ht_par .. " -cooxtpipes -coo "
```

There are some functions to ease access to the settings:

set_settings{parameters} overwrite settings with values from a passed table

settings_add{parameters} add values to the current settings

filter_settings "filter name" {parameters} set settings for a filter

get_filter_settings(name) get settings for a filter

Using these functions, it is possible to simplify the settings for the ODT format:

```
settings_add {
  tex4ht_sty_par =",ooffice",
  tex4ht_par = " ooffice/! -cmozhtf",
  t4ht_par = " -cooxtpipes -coo "
}
```

Settings for filters and extensions can be set using `filter_settings`:

```
filter_settings "test" {
  hello = "world"
}
```

These settings can be read in the extensions and filters using `get_filter_settings`:

```
function test(input)
   local options = get_filter_settings("test")
   print(options.hello)
   return input
end
```

3.6 List of available settings for filters and extensions.

These settings may be set using `filter_settings` function.

3.6.1 The `tidy` extension

options command line options for the `tidy` command. Default value is `-m -utf8 -w 512 -q`.

3.6.2 The `fixinlines` dom filter

inline_elements table of inline elements which shouldn't be direct descendants of the `body` element. The element names should be table keys, the values should be true.

Example

```
filter_settings "fixinlines" {inline_elements = {a = true, b = true}}
```

3.6.3 The `joincharacters` dom filter

charelements table of elements which should be joined if several instances with the same value of `class` attribute are side by side.

Example

```
filter_settings "joincharacters" { charclases = { span=true, mn = true}}
```

3.6.4 The `mathjaxnode` filter

options command line options for the `mjpage` command. Default value is `--output CommonHTML`

Example

```
filter_settings "mathjaxnode" {
  options="--output SVG --font Neo-Euler"
}
```

cssfilename `mjpage` puts some CSS code into the HTML pages. `mathjaxnode` extracts this information and saves it to a standalone CSS file. Default CSS filename is `mathjax-chtml.css`

fontdir directory with MathJax font files. This option enables use of local fonts, which is usefull in Epub conversion, for example. The font directory should be sub-directory of the current directory. Only TeX font is supported at the moment.

Example

```
filter_settings "mathjaxnode" {
  fontdir="fonts/TeX/woff/"
}
```

3.6.5 The `aeneas` filter

skip_elements List of CSS selectors that match elements which shouldn't be processed. Default value: `{ "math", "svg"}`.

id_prefix prefix used in the ID attribute forming.

sentence_match Lua pattern used to match a sentence. Default value: `"([^%.^%?^!]*)([%.%?!]?)"`.

3.6.6 The `staticsite` filter and extension

site_root directory where generated files should be copied.

map table where keys are patterns that match filenames, value contains destination directoryfor matched files, relative to the `site_root` (it is possible to use `..` to swich to parent directory).

file_pattern pattern used for filename generation. It is possible to use string templates and format strings for `os.date` function. Default value of `%Y-%m-%d-${input}` creates names in the form of `YYYY-MM-DD-file_name`.

header table with variables to be set in the YAML header in HTML files. If the table value is a function, it is executed with current parameters and HTML page DOM object as arguments.

Example:

```
local outdir = os.getenv "blog_root"

filter_settings "staticsite" {
  site_root = outdir,
  map = {
    [".css$"] = "../css/"
  },
  header = {
     layout="post",
     date = function(parameters, dom)
       return os.date("!%Y-%m-%d %T", parameters.time)
     end
  }
}
```

4 Configuration file

It is possible to globally modify the build settings using the configuration file. New compilation commands can be added, extensions can be loaded or disabled and settings can be set.

4.1 Location

The configuration file can be saved either in `$HOME/.config/make4ht/config.lua` or in `.make4ht` in the current directory or it's parents (up to `$HOME`).

4.2 Additional commands

There are two additional commands:

`Make:enable_extension(name)` require extension

`Make:disable_extension(name)` disable extension

4.3 Example

The following configuration add support for the `biber` command, requires `common_domfilters` extension and requires MathML output for math.

```
Make:add("biber", "biber ${input}")
Make:enable_extension "common_domfilters"
settings_add {
  tex4ht_sty_par =",mathml"
}
```

5 Command line options

```
make4ht - build system for tex4ht
Usage:
make4ht [options] filename ["tex4ht.sty op." "tex4ht op."
     "t4ht op" "latex op"]
-b,--backend (default tex4ht) Backend used for xml generation.
     possible values: tex4ht or lua4ht
-c,--config (default xhtml) Custom config file
-d,--output-dir (default "")  Output directory
-e,--build-file (default nil)  If build file name is different
     than `filename`.mk4
-f,--format  (default nil)  Output file format
-l,--lua  Use lualatex for document compilation
-m,--mode (default default) Switch which can be used in the makefile
-n,--no-tex4ht  Disable dvi file processing with tex4ht command
-s,--shell-escape Enables running external programs from LaTeX
-u,--utf8  For output documents in utf8 encoding
-v,--version  Print version number
-x,--xetex Use xelatex for document compilation
<filename> (string) Input file name
```

You can still invoke `make4ht` in the same way as `htlatex`:

```
make4ht filename "customcfg, charset=utf-8" "-cunihtf -utf8" "-dfoo"
```

Note that this will not use `make4ht` routines for output directory making and copying. If you want to use them, change the line above to:

```
make4ht -d foo filename "customcfg, charset=utf-8" "-cunihtf -utf8"
```

This call has the same effect as the following:

```
make4ht -u -c customcfg -d foo filename
```

Output directory doesn't have to exist, it will be created automatically. Specified path can be relative to current directory, or absolute:

```
make4ht -d use/current/dir/ filename
make4ht -d ../gotoparrentdir filename
make4ht -d ~/gotohomedir filename
make4ht -d c:\documents\windowspathsareworkingtoo filename
```

6 Troubleshooting

6.1 Incorrect handling of command line arguments for `tex4ht`, `t4ht` or `latex`

Sometimes, you may get a similar error:

```
make4ht:unrecognized parameter: i
```

It may be caused by a following make4ht invocation:

```
make4ht hello.tex "customcfg,charset=utf-8" "-cunihtf -utf8" -d foo
```

The command line option parser is confused by mixing options for make4ht and tex4ht in this case and tries to interpret the -cunihtf -utf8, which are options for tex4ht command as make4ht options. To fix that, you can either move the -d foo directly after make4ht command:

```
make4ht -d foo hello.tex "customcfg,charset=utf-8" "-cunihtf -utf8"
```

Another option is to add a space before tex4ht options:

```
make4ht hello.tex "customcfg,charset=utf-8" " -cunihtf -utf8" -d foo
```

The former way is preferable, though.

7 License

8 Changelog

- 2018/05/03
  - renamed latexmk extension to latexmk_build, due to clash in TL
- 2018/04/18
  - staticsite extension:
    - make YAML header configurable
    - set the time and updated headers
  - don't override existing tables in filter_settings
- 2018/04/17
  - done first version of staticsite extension
- 2018/04/16
  - check for Git repo in the Makefile, don't run Git commands outside of repo
- 2018/04/15
  - added staticsite filter

  - working on `staticsite` extension

- 2018/04/13
  - use `ipairs` instead of `pairs` to traverse lists of images and image match functions
  - load extensions in the correct order

- 2018/04/09
  - released version `0.2`
  - disabled default loading of `common_domfilters` extension

- 2018/04/06
  - added `Make:enable_extension` and `Make:disable_extension` functions
  - documented the configuration file

- 2018/03/09
  - load the configuration file before extensions

- 2018/03/02
  - Aeneas execution works
  - Aeneas documentation
  - added support for `.make4ht` configuration file

- 2018/02/28
  - Aeneas configuration file creation works

- 2018/02/22
  - fixed bug in `fixinlines` DOM filter

- 2018/02/21
  - added Aeneas domfilter
  - fixed bugs in `joincharacters` DOM filter

- 2018/02/20
  - fixed bug in `joincharacters` DOM filter
  - make `woff` default font format for `mathjaxnode`
  - added documentation for `mathjaxnode` settings

- 2018/02/19
  - fixed bug in filter loading
  - added `mathjaxnode` extension
- 2018/02/15
  - use HTML5 as a default format
  - use `common_domfilters` implicitly for the XHTML and HTML5 formats
- 2018/02/12
  - added `common_domfilters` extension
  - documented DOM filters
- 2018/02/12
  - handle XML parsing errors in the DOM handler
  - enable extension loading in Formatters
- 2018/02/11
  - fixed Tidy extension output to support LuaXML
  - fixed white space issues with `joincharacters` DOM filter
- 2018/02/09
  - fixed issues with the Mathjax filter
  - documented basic info about thd DOM filters
  - DOM filter optimalizations
- 2018/02/08
  - make Tidy extension configurable
  - documented filter settings
- 2018/02/07
  - added filter for Mathjax-node
- 2018/02/06
  - created DOM filter function
  - added DOM filter for spurious inlinine elements
- 2018/02/03

  - added settings handling functions
  - settings made available for extensions and filters

- 2017/12/08

  - fixed the `mk4` build file loading when it is placed in the current working dir and another one with same filename somewhere in the TEXMF tree.

- 2017/11/10

  - Added new filter: `svg-height`. It tries to fix height of some of the images produced by `dvisvgm`

- 2017/10/06

  - Added support for output format selection. Supported formats are `xhtml`, `html5` and `odt`
  - Added support for extensions

- 2017/09/10

  - Added support for Latexmk
  - Added support of `math` library and `tonumber` function in the build files

- 2017/09/04

  - fixed bug caused by the previous change – the –help and –version didn't work

- 2017/08/22

  - fixed the command line option parsing for `tex4ht`, `t4ht` and `latex` commands
  - various grammar and factual fixes in the documentation

- 2017/04/26

  - Released version `v0.1c`

- 2017/03/16

  - check for `TeX capacity exceeded` error in the LaTeX run.

- 2016/12/19

  - use full input name in `tex_file` variable. This should enable use of files without `.tex` extension.

- 2016/10/22
  - new command available in the build file: `Make:add_file(filename)`. This enables filters and commands to register files to the output.
  - use ipairs instead of pairs for traversing files and executing filters. This should ensure correct order of executions.

- 2016/10/18
  - new filter: replace colons in `id` and `href` attributes with underscores

- 2016/01/11
  - fixed bug in loading documents with full path specified

- 2015/12/06 version 0.1b
  - modifed lapp library to recognize `--version` and
  - added `--help` and `--version` command line options

- 2015/11/30
  - use `kpse` library for build file locating

- 2015/11/17
  - better `-jobname` handling

- 2015/09/23 version 0.1a
  - various documentation updates
  - `mozhtf` profile for unicode output is used, this should prevent ligatures in the output files

- 2015/06/29 version 0.1
  - major README file update

- 2015/06/26
  - added Makefile
  - moved INSTALL instructions from README to INSTALL