The make4ht build system

Michal Hoftich*

Version v0.3b
2019-11-03

Contents

*<michal.h21@gmail.com>

1 Introduction

make4ht is a build system for TEX4ht, TEX to XML converter. It provides a command line tool that drives the conversion process. It also provides a library that can be used to create customized conversion tools. An example of such a tool is tex4ebook, a tool for conversion from TEX to ePub and other e-book formats.

The basic conversion from LATEX to HTML using make4ht can be executed using the following command:

```
$ make4ht filename.tex
```

It will produce a file named `filename.html` if the compilation goes without fatal errors.

2 Command line options

```
make4ht - build system for TeX4ht
Usage:
make4ht [options] filename ["tex4ht.sty op." "tex4ht op."
     "t4ht op" "latex op"]
-a,--loglevel (default status) Set log level.
            possible values: debug, info, status, warning, error, fatal
-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 filename is different
     than `filename`.mk4
-f,--format  (default nil)  Output file format
-j,--jobname (default nil)  Set the jobname
-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
-x,--xetex Use xelatex for document compilation
-v,--version  Print version number
<filename> (string) Input filename
```

It is still possible to invoke `make4ht` in the same way as is invoked `htlatex`:

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

Note that this will not use `make4ht` routines for the output directory handling. See section 3.3 for more information about this issue. To use these routines, change the previous listing 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 the current directory, or absolute:

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

The short options that don't take parameters can be collapsed:

```
$ make4ht -ulc customcfg -d foo filename
```

To pass output from the other commands to `make4ht` use the `-` character as a filename. It is best to use this feature together with the `--jobname` or `-j` option.

```
$ cat hello.tex | make4ht -j world -
```

By default, `make4ht` tries to be quiet, so it hides most of the command line messages and the output from the executed commands. It will display only status messages, warnings and errors. The logging level can be selected using the `--loglevel` or `-a` options. If the compilation fails, it may be useful to display more information using the `info` or `debug` levels.

```
$ make4ht -a debug faulty.tex
```

3 Why `make4ht`? – `htlatex` issues

TEX4ht system supports several output formats, most notably `XHTML`, `HTML 5` and `ODT`, but it also supports `TEI` or `Docbook`.

The conversion can be invoked using several scripts, which are distributed with TEX4ht. They differ in parameters passed to the underlying commands.

These scripts invoke LATEX or Plain TEX with special instructions to load the `tex4ht.sty` package. The TEX run produces a special `DVI` file that contains the code for the desired output format. The produced `DVI` file is then processed using the `tex4ht` command, which in conjunction with the `t4ht` command produces the desired output files.

3.1 Passing command line arguments

The basic conversion script provided by TEX4ht system 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 the HTML 5 output in UTF-8 encoding:

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

make4ht can simplify it:

```
$ make4ht -u filename.tex
```

The -u option requires the UTF-8 encoding. HTML 5 is used as the default output format by make4ht.

More information about the command line arguments can be found in section 2.

3.2 Compilation sequence

htlatex has a fixed compilation order and a hard-coded number of LaTeX invocations.

It is not possible to execute additional commands during the compilation. When we want to run a program that interacts with LaTeX, such as Makeindex or Bibtex, we have two options. The first option is to create a new script based on htlatex and add the wanted commands to the modified script. The second option is to execute htlatex, then the additional and then htlatex again. The second option means that LaTeX will be invoked six times, as each call to htlatex executes three calls to LaTeX. This can lead to significantly long compilation times.

make4ht provides a solution for this issue using a build file, or extensions. These can be used for interaction with external tools.

make4ht also provides compilation modes, which enables to select commands that should be executed using a command line option.

There is a built-in draft mode, which invokes LaTeX only once, instead of the default three invocations. It is useful for the compilations of the document before its final stage, when it is not important that all cross-references work. It can save quite a lot of the compilation time:

```
$ make4ht -um draft filename.tex
```

More information about the build files can be found in section 5.

3.3 Handling of the generated files

There are also issues with the behavior of the t4ht application. It reads the .lg file generated by the tex4ht command. This file contains information about the generated files, CSS instructions, calls to the external applications, instructions for image conversions, etc.

t4ht can be instructed to copy the generated files to an output directory, but it doesn't preserve the directory structure. When the images are placed in a

subdirectory, they will be copied to the output directory, losing the directory structure. 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
```

3.4 Image conversion and post-processing of the generated files

TEX4ht can convert parts of the document to images. This is useful for diagrams or complicated math, for example.

By default, the image conversion is configured in a `.env` file. It has a bit of strange syntax, with operating system dependent rules. `make4ht` provides simpler means for the image conversion in the build files. It is possible to change the image conversion parameters without a need to modify the `.env` file. The process is described in section 5.4.

It is also possible to post-process the generated output files. The post-processing can be done either using external programs such as `XSLT` processors and `HTML Tidy` or using `Lua` functions. More information can be found in section 5.3.

4 Output file formats and extensions

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

- `xhtml`
- `html5`
- `odt`
- `tei`
- `docbook`

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

4.1 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:

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`, `joincharacters`, and `tablerows`.

dvisvgm_hashes efficient generation of SVG pictures using Dvisvgm. It can utilize multiple processor cores and generates only changed images.

join_colors load the `joincolors` domfilter for all HTML files.

latexmk_build use Latexmk for the LaTeX compilation.

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

odttemplate it automatically loads the `odttemplate` filter (page 11).

preprocess_input compilation of the formats supported by Knitr (`.Rnw`, `.Rtex`, `.Rmd`, `.Rrst`) and also Markdown and reStructuredText formats. It requires R + Knitr installation, it requires also Pandoc for formats based on Markdown or reStructuredText.

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

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

5 Build files

`make4ht` supports build files. These are `Lua` scripts that can adjust the build process. They 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`. It is possible to select a 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 the `tex4ht.sty` package loaded. In this example, it will be executed only once. After the compilation, the `tidy` command is executed on the output `HTML` files.

Note that it is not necessary to call `tex4ht` and `t4ht` commands explicitly in the build file, they are called automatically.

5.1 User commands

It is possible to add more commands like `Make:htlatex` using the `Make:add` command:

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

This defines the `name` command, which can be then executed using `Make:name()` command in the build file.

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

The defined command receives a table with settings as a parameter at the call time. The default settings are provided by `make4ht`. Additional settings can be declared in the `Make:add` commands, user can also override the default settings when the command is executed in the build file:

```
Make:name({hello="world"})
```

More information about settings, including the default settings provided by `make4ht`, can be found in section 5.6 on page 13.

5.1.1 The `command` function

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

```
Make:add("text", "echo hello, input file: ${input}")
```

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

5.1.2 The `settings table` table

The `settings table` parameter is optional. If it is present, it should be a table with new settings available in the command. It can also override the default `make4ht` settings for the defined command.

```
Make:add("sample_function", function(params)
  for k, v in pairs(params) do
    print(k..": "..v)
  end, {custom="Hello world"}
)
```

5.1.3 Repetition

The `repetition` parameter specifies the maximum 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 when they are included in the build file, as they are called by `make4ht` by default. Because these commands allow only one `repetition`, the second execution is blocked.

5.1.4 Expected exit code

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

```
Make:add("biber", "biber ${input}", {correct_exit=0})
```

Commands that execute lua functions can return the numerical values using the `return` statement.

This mechanism isn't used for TEX, because it doesn't differentiate between fatal and non-fatal errors. It returns the same exit code in all cases. Because of this, log parsing is used for a fatal error detection instead. Error code value `1` is returned in the case of a 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.

5.2 Provided commands

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

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

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

`Make:t4ht` Create the CSS file and generate images.

`Make:biber` Process bibliography using the `biber` command.

`Make:bibtex` Process bibliography using the `bibtex` command.

`Make:xindy` Generate index using Xindy index processor.

`Make:makeindex` Generate index using the Makeindex command.

`Make:xindex` Generate index using the Xindex command.

5.3 File matches

Another type of action that can be specified in the build file is `Make:match`. It can be used to post-process the generated files:

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

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

The `Make:match` action tests output filenames using a `Lua` pattern matching function.

It executes a command or a function, specified in the second argument, on files whose filenames match the pattern.

The commands to be executed can be specified as strings. They can contain `${var_name}` placeholders, which are replaced with corresponding variables from the `settings` table. The templating system was described in subsection 5.1.1. There is an additional variable available in this table, called `filename`. It contains the name of the current output file.

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

```
Make:match("html$", function(filename, settings)
  print("Post-processing file: ".. filename)
  print("Available settings")
  for k,v in pairs(settings)
    print(k,v)
  end
  return true

  end)
```

Multiple post-processing actions can be executed on each filename. The Lua action functions can return an exit code. If the exit code is false, the execution of the post-processing chain for the current file will be terminated.

5.3.1 Filters

To make it easier to post-process the generated files using the `match` actions, `make4ht` provides a filtering mechanism thanks to the `make4ht-filter` module.

The `make4ht-filter` module returns a function that can be used for the filter chain building. Multiple filters can be chained into a pipeline. Each filter can modify the string that is passed to it from the previous filters. The changes are then saved to the processed file.

Several built-in filters are available, it is also possible to create new ones.

Example that use only the built-in filters:

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

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

Example with custom filter:

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

Built-in filters are the following:

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 translates these underscores to `<hr>` elements

entites convert prohibited named entities to numeric entities (only ` ` currently).

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 a validation error.

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

odttemplate use styles from another `ODT` file serving as a template in the current document. It works for the `styles.xml` file in the `ODT` file. During the compilation, this file is named as `\jobname.4oy`.

staticsite create HTML files in a 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.

5.3.2 DOM filters

DOM filters are variants of filters that 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 synchronization.

collapsetoc collapse table of contents to contain only top-level sectioning level and sections on the current page.

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. This DOM filter supersedes the `cleanspan` filter.

joincolors many `<span>` elements with unique `id` attributes are created when LaTeX colors are being used in the document. A CSS rule is added for each of these elements, which may result in substantial growth of the CSS file. This filter replaces these rules with a common one for elements with the same color value.

odtimagesize set correct dimensions for images in the ODT format. It is loaded by default for the ODT output.

odtpartable resolve tables nested inside paragraphs, which is invalid in the ODT format.

tablerows remove spurious rows from HTML tables.

t4htlinks fix hyperlinks in the ODT format.

5.4 Image conversion

It is possible to convert parts of the LaTeX input as pictures. It can be used for preserving the appearance of math or diagrams, for example.

These pictures are stored in a special `DVI` file, which can be processed by a `DVI` to image commands, such as `dvipng` or `dvisvgm`.

This conversion is normally configured in the `tex4ht.env` file. This file is system dependent and it has quite an unintuitive syntax. The configuration is processed by the `t4ht` application and the conversion command is called for all pictures.

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

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

`Make:image` takes two parameters, a `Lua` pattern to match the image name, and the action.

Action can be either a string template with the conversion command or a function that takes a table with parameters as an argument.

There are three parameters:

- `output` - output image filename
- `source` - `DVI` file with the pictures
- `page` - page number of the converted image

5.5 The `mode` variable

The `mode` variable available in the build process 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 the `draft` mode:

```
make4ht -m draft filename
```

5.6 The `settings` table

It is possible to access the parameters outside commands, file matches and image conversion functions. For example, to convert the document to the `OpenDocument Format (ODT)`, the following settings can be used. They are 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 "
```

(Note that it is possible to use the `--format odt` option which is superior to the previous code. This example is intended just as an illustration)

There are some functions to simplify 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

For example, it is possible to simplify the sample from the previous code listings:

```
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 retrieved in the extensions and filters using the `get_filter_settings` function:

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

5.6.1 Default settings

The default parameters are the following:

`htlatex` used TEX engine

`input` content of `\jobname`, see also the `tex_file` parameter.

`interaction` interaction mode for the TEX engine. The default value is `batchmode` to suppress user input on compilation errors. It also suppresses most of the TEX compilation log output. Use the `errorstopmode` for the default behavior.

`tex_file` input TEX filename

`latex_par` command line parameters to the TEX engine

`packages` additional LATEX code inserted before `\documentclass`. Useful for passing options to packages used in the document or to load additional packages.

`tex4ht_sty_par` options for `tex4ht.sty`

`tex4ht_par` command line options for the `tex4ht` command

`t4ht_par` command line options for the `t4ht` command

`outdir` the output directory

`correct_exit` expected `exit code` from the command. The compilation will be terminated if the exit code of the executed command has a different value.

6 Configuration file

It is possible to globally modify the build settings using the configuration file. It is a special version of a build file where the global settings can be set.

Common tasks for the configuration file can be a declaration of the new commands, loading of the default filters or specification of a default build sequence.

One additional functionality not available in the build files are commands for enabling and disabling of extensions.

6.1 Location

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

6.2 Additional commands

There are two additional commands:

`Make:enable_extension(name)` require extension

`Make:disable_extension(name)` disable extension

6.3 Example

The following example of the configuration file adds 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"
}
```

7 List of available settings for filters and extensions.

These settings may be set using `filter_settings` function in a build file or in the `make4ht` configuration file.

7.1 Indexing commands

The indexing commands (like `xindy` or `makeindex`) use some common settings.

idxfile name of the `.idx` file. Default value is `\jobname.idx`.

indfile name of the `.ind` file. Default value is the same as `idxfile` with the file extension changed to `.ind`.

Each indexing command can have some additional settings.

7.1.1 The `xindy` command

encoding text encoding of the `.idx` file. Default value is `utf8`.

language index language. Default language is English.

modules table with names of additional `Xindy` modules to be used.

7.1.2 The `makeindex` command

options
: additional command line options for the Makeindex command.

7.1.3 The `xindex` command

options
: additional command line options for the Xindex command.

language document language

7.2 The `tidy` extension

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

7.3 The `collapsetoc` dom filter

`toc_query` CSS selector for selecting the table of contents container.

`title_query` CSS selector for selecting all elements that contain the section ID attribute.

`toc_levels` table containing a hierarchy of classes used in TOC

Default values:

```
filter_settings "collapsetoc" {
  toc_query = ".tableofcontents",
  title_query = ".partHead a, .chapterHead a, .sectionHead a, .subsectionHead a",
  toc_levels = {"partToc", "chapterToc", "sectionToc", "subsectionToc", "subsubsectionToc"}
}
```

7.4 The `fixinlines` dom filter

inline_elements table of inline elements that 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}}
```

7.5 The `joincharacters` dom filter

charclasses table of elements that should be concatenated when two or more of such elements with the same value of the `class` attribute are placed one after another.

Example

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

7.6 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 the `mjpage` command 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 the use of local fonts, which is useful in the conversion to ePub, 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/"
}
```

7.7 The `staticsite` filter and extension

site_root directory where generated files should be copied.

map a hash table where keys contain patterns that match filenames and values contain destination directory for the matched files. The destination directories are relative to the `site_root` (it is possible to use `..` to switch to a parent directory).

file_pattern a pattern used for filename generation. It is possible to use string templates and format strings for `os.date` function. The default pattern `%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
  }
}
```

7.8 The `dvisvgm_hashes` extension

options command line options for Dvisvgm. The default value is `-n --exact -c 1.15,1.15`.

cpu_cnt the number of processor cores used for the conversion. The extension tries to detect the available cores automatically by default.

parallel_size the number of pages used in each Dvisvgm call. The extension detects changed pages in the DVI file and constructs multiple calls to Dvisvgm with only changed pages.

scale SVG scaling.

7.9 The `odttemplate` filter and extension

template filename of the template `ODT` file

`odttemplate` can also get the template filename from the `odttemplate` option from `tex4ht_sty_par` parameter. It can be set using the following command line call:

```
make4ht -f odt+odttemplate filename.tex "odttemplate=template.odt"
```

7.10 The `aeneas` filter

skip_elements List of CSS selectors that match elements that 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: `"([^%.^%?^!]*)([%.%?!]?)"`.

7.11 The `make4ht-aeneas-config` package

Companion for the `aeneas` DOM filter is the `make4ht-aeneas-config` plugin. It can be used to write the 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, a `SMIL` file is created. It is assumed that there is an audio file in the `mp3` format, named as the TEX file. It is possible to use different formats and filenames using mapping.

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

7.11.1 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 below.

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

id_sort sorting of id attributes. The default value is `numeric`.

id_regex regular expression to parse the id attributes.

sub_format generated subtitle format. The default value is `smil`.

7.11.2 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 a 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 the `map` parameter.

Example:

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

Table keys are the configured filenames. 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 a corresponding audio file.

Filenames of the subfiles correspond to the chapter numbers, so they are not stable when a new chapter is added. It is possible to request filenames derived from the chapter titles using the `sec-filename` option for `tex4ht.sty`.

7.11.3 Available `map` options

audio_file the corresponding audio file

sub_file name of the generated subtitle file

The following options are the same as their counterparts 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

7.11.4 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
```

8 Troubleshooting

8.1 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. It tries to interpret the `-cunihtf -utf8`, which are options for the `tex4ht` command, as `make4ht` options. To fix that, try to move the `-d foo` directly after the `make4ht` command:

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

Another option is to add a space before the tex4ht options:

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

The former way is preferable, though.

8.2 Filenames containing spaces

tex4ht command cannot handle filenames containing spaces. to fix this issue, make4ht replaces spaces in the input filenames with underscores. The generated XML filenames use underscores instead of spaces as well.

8.3 Filenames containing non-ASCII characters

The odt output doesn't support accented filenames, it is best to stick to ASCII characters in filenames.

9 License

10 Changelog

- 2019/11/03
  - version 0.3b
  - use make4ht-ext- prefix for extensions to prevent filename clashes with corresponding filters.
- 2019/11/01
  - version 0.3a released.
  - added make4ht- prefix to all extensions and formats
  - removed the unused mathjaxnode.lua file.
- 2019/11/01
  - version 0.3 released.
  - added Make:makeindex, Make:xindex and Make:bibtex commands.
- 2019/10/25
  - modified the Make:xindy command to use the indexing mechanism.

- 2019/10/24
  - added functions for preparing and cleaning of the index files in `make4ht-indexing.lua`.
- 2019/10/23
  - replaced `os.execute` function with `mkutils.execute`. It uses the logging mechanism for the output.
  - finished transforming of filters, extensions and formats to the logging system.
- 2019/10/22
  - added `tablerows` domfilter.
  - added the `tablerows` domfilter to the `common_domfilters` extension.
  - converted most of the filters to use the logging mechanism.
- 2019/10/20
  - added `status` log level.
- 2019/10/18
  - converted most print commands to use the logging mechanism.
  - added `output` log level used for printing of the commands output.
- 2019/10/17
  - added `--loglevel` CLI parameter.
  - added logging mechanism.
  - moved `htlatex` related code to `make4ht-htlatex.lua` from `mkutils.lua`
- 2019/10/11
  - added `xindy` settings.
  - added simple regular expression to detect errors in the log file, because log parsing can be slow.
- 2019/10/09
  - added the `interaction` parameter for the `htlatex` command. The default value is `batchmode` to suppress the user input on errors, and to suppress full log output to the terminal.
  - added the `make4ht-errorlogparser` module. It is used to parse errors in the `htlatex` run unless `interaction` is set to `errorstopmode`.

- 2019/10/08
  - set up Github Actions pipeline to compile the documentation to HTML and publish it at https://www.kodymirus.cz/make4ht/make4ht-doc.html.
- 2019/10/07
  - don't move the `common_domfilters` extension to the first place in the file matches pipeline. We may want to run `tidy` or regex filters first, to fix XML validation errors.
- 2019/10/04
  - added HTML documentation.
- 2019/09/27
  - don't convert Latin 1 entities to Unicode in the `entities_to_unicode` extension.
- 2019/09/20
  - fixed bugs in the temporary directory handling for the ODT output.
- 2019/09/13
  - added `preprocess_input` extension. It enables compilation of formats supported by Knitr (`.Rnw`, `.Rtex`, `.Rmd`, `.Rrst`) and also Markdown and reStructuredText formats.
- 2019/09/12
  - added support for the ODT files in `common_domfilters` extension.
  - renamed `charclases` option for the `joincharacters` DOM filter to `charclasses`.
  - don't execute the `fixentities` filter before Xtpipes, it makes no sense.
- 2019/09/11
  - added support for Biber in the build files.
- 2019/08/28
  - added support for input from `stdin`.
- 2019/08/27
  - fixed `-jobname` detection regex.

  - added function `handle_jobname`.
  - added the `--jobname` command line option.

- 2019/08/26

  - quote file names and paths in `xtpipes` and `tidy` invocation.

- 2019/08/25

  - the issue tracker link in the help message is now configurable.
  - fixed bug in the XeTeX handling: the `.xdv` argument for `tex4ht` wasn't used if command line arguments for `tex4ht` were present.

- 2019/07/03

  - new DOM filter: `odtpartable`. It fixes tables nested in paragraphs in the ODT format.

- 2019/06/13

  - new DOM extension: `collapsetoc`.

- 2019/05/29

  - new module: `make4ht-indexing` for working with index files.

- 2019/05/24

  - version 0.2g released
  - fixed failing `dvisvgm_hashes` extension on Windows.

- 2019/05/02

  - fixed infinite loop bug in the `dvisvgm_hashes` extension.

- 2019/04/09

  - `make4ht-joincolors` fix: remove the hash character from the color name. This caused issues with colors specified in the hexadecimal format.

- 2019/04/02

  - `dvisvgm_hashes` fix: update also the lgfile.images table with generated filenames, in order to support tex4ebook

- 2019/04/01

  - fixed bug in dvisvgm_hashes extension: didn't check for table index existence in string concenation

- 2019/03/21
  - version 0.2f released
- 2019/03/15
  - check for the image dimensions existence in the odtimagesize domfilter.
- 2019/03/13
  - don't use odtimagesize domfilter in the ODT format, the issue it fixes had been resolved in tex4ht.
- 2019/03/08
  - use %USERPROFILE for home dir search on Windows.
- 2019/01/28
  - added joincolors domfilter and join_colors extension. It can join CSS rules created for the LaTeX colors and update the HTML file.
- 2019/01/22
  - version 0.2e released
  - updated the odttemplate filter. It will use styles from the generated ODT file that haven't been present in the template file.
- 2019/01/10
  - version 0.2d released
- 2019/01/05
  - added docbook and tei output formats.
- 2018/12/19
  - new library: make4ht-xtpipes.lua. It contains code for xtpipes handling.
  - moved Xtpipes handling code from formats/odt.lua.
- 2018/12/18

  - new filter: `odttemplate`. It can be used for replacing style in a generated `ODT` file by a style from another existing `ODT` file.
  - new extension: `odttemplate`. Companioning extension for filter with the same name.
  - fixed bug in `make4ht-filters.lua`: the parameters table haven't been passed to filters.

- 2018/12/17

  - fixed extension handling. The disabling from the command line didn't take precedence over extensions enabled in the config file. Extensions also could be executed multiple times.

- 2018/11/08

  - removed replacing newlines by blank strings in the `joincharacters` domfilter. The issue it fixed doesn't seem to exist anymore, and it ate spaces sometimes.

- 2018/11/01

  - added `t4htlinks` domfilter
  - fixed the `xtpipes` and `filters` execution order in the `ODT` format

- 2018/10/26

  - fixed ODT generation for files that contains special characters for Lua string patterns
  - replace non-breaking spaces with entities. It caused issues in LO

- 2018/10/18

  - fixed the executable installation

- 2018/09/16

  - added the `scale` option for `dvisvgm_hashes` extension

- 2018/09/14

  - require the `-dvi` option with `latexmk_build` extension

- 2018/09/12

  - added `xindy` command for the build file

- 2018/09/03

  - expanded the `--help` option

- 2018/08/27
  - added `odtimagesize` domfilter
  - load `odtimagesize` by default in the ODT format

- 2018/08/23
  - released version 0.2c

- 2018/08/21
  - added processor core detection on Windows
  - make processor number configurable
  - updated the documentation.

- 2018/08/20
  - added `dvisvgm_hashes` extension

- 2018/07/03
  - create the `mimetype` file to achieve the ODT file validity

- 2018/07/02
  - disabled conversion of XML entities for &, < and > characters back to Unicode, because it breaks XML validity

- 2018/06/27
  - fixed root dir detection

- 2018/06/26
  - added code for detection of TeX distribution root for Miktex and TL

- 2018/06/25
  - moved call to `xtpipes` from `t4ht` to the `ODT` format drives. This should fix issues with path expansion in `tex4ht.env` in TeX distributions.

- 2018/06/22
  - added `mkutils.find_zip` function. It detects `zip` or `miktex-zip` executables

- 2018/06/19

  - added new filter: `entities-to-unicode`. It converts XML entites for Unicode characters back to Unicode.
  - execute `entities-to-unicode` filter on text and math files in the ODT output.

- 2018/06/12
  - added support for direct `ODT` file packing

- 2018/06/11
  - new function available for formats, `format.modify_build`
  - function `mkutils.delete_dir` for directory removal
  - function `mkutils.mv` for file moving
  - started on packing of the `ODT` files directly by the format, instead of `t4ht`

- 2018/06/08
  - added support for filenames containing spaces
  - added support for filenames containing non-ascii characters
  - don't require sudo for the installation, let the user to install symbolic links to `$PATH`

- 2018/05/03
  - released version `0.2b`
  - bug fix: use only `load` function in `Make:run`, in order to support a local environment.

- 2018/05/03
  - released version `0.2a`
  - 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