Offline Reading

Exporting blog posts from HTML to ePub and PDF

Willem L. Middelkoop
Jan. 19, 2026

This year marks the 10th anniversary of my blog. What started as an experiment has evolved into over 170 stories, averaging around 60 thousand pageviews a month. I recently started translating posts into Dutch, German, and Spanish, expanding the reach. And now I am introducing support for offline reading by making all posts available as ePub and PDF downloads.

Read willem.com posts on your favourite e-reader (like Kobo or reMarkable)

Why Offline?

Enjoy distraction free reading on e-ink devices like a Kobo or reMarkable which are kind to your eyes. Focus on the articles wherever and whenever you want as they are available as downloads that work offline and DRM-free. All willem.com ePubs and PDFs contain high quality images and feature beautiful resizable typography. I made a handy overview of ePub and PDF downloads here:

Download ePub and PDF

How it works

The willem.com website runs on the Lemmid content management system that is a static site generator, outputting optimised HTML suitable for high volume traffic and international distribution. I extended the NodeJS code to automatically export to ePub and PDF by using simplified HTML as input. See the source code snippets below: for **ePub** I use the epub-gen module; for **PDF** I use pandoc and LaTeX. I made a task manager that queues output jobs for individual translations and file formats, spreading computational load on the backend.

Conclusion

The reactions I have received on my posts come from all over the world and they inspire me to keep writing. With ePub and PDF support I hope to expand my reach even further by enabling more ways to read the articles. Enjoy!

```
Publisher.prototype.performOutputJobEpub = function(job, callback) {
   // console.log('Publisher: performOutputJobEpub '+job.instance.id+' as '+job.type);
   var instance = job.instance;
   var language = job.language;
   var languageSuffix = language.toUpperCase();
   var epubFilename = instance['slug' + languageSuffix] + '.epub';
   this.base.isFileNewerThanTimestamp(instance['absolutePath' + languageSuffix] + epubFilename, instance.lastUpdated, function(isNewer) {
      if (isNewer) {
         callback(); // no action needed
      } else {
         // the ePub needs to be generated
         const heroImagePath = (instance['image' + languageSuffix])
            ? this.base.resultPath + instance['image' + languageSuffix].url.w1000
            : this.base.resultPath + '/global/icon.png';
         // Localised strings for the Table of Contents
         const tocTitles = {
            en: "Table of Contents",
            nl: "Inhoudsopgave",
            de: "Inhaltsverzeichnis",
            es: "Tabla de Contenidos"
         };
         const fullTitle = instance['title' + languageSuffix] + ': ' + instance['subTitle' + languageSuffix];
         // the inputHtml is based on feedHtml which uses absolute URL's, since
         // we're processing locally on the filesystem into epubs, we're removing
         // online (image) links, transforming them in absolute filesystem paths,
         // we also remove the anti cache parameter ?=LASTUPDATED
         const inputHtml = '<h4>' + instance['dateFormatted' + languageSuffix] + '</h4>\n' +
               this.base.replaceAllInString(
                 this.base.replaceAllInString(instance['feedHtml' + languageSuffix],
                     'https://' + this.base.rootFolderName + instance['relativePath' + languageSuffix],
                     instance['absolutePath' + languageSuffix]),
                 '?u=' + instance.lastUpdated, '');

         const options = {
            title: fullTitle,
            author: this.authorName,
            publisher: this.base.rootFolderName,
            lang: language,
            cover: heroImagePath,
            tocTitle: tocTitles[language] || tocTitles['en'], tempDir: this.rootPath + '/tmp',
            content: this.base.splitHtmlIntoChapters(inputHtml, fullTitle),
            appendChapterTitles: true,
            verbose: this.isDebug
         };
         const outputPath = instance['absolutePath' + languageSuffix] + epubFilename;
         // Generate the EPUB using promise .then/.catch for callback hook.
         new Epub(options, outputPath).promise .then(() => {
            console.log(`Publisher: Successfully generated EPUB for ${instance.id}`);
            if (callback)
               callback();
         }) . catch ((error) => {
            console.error(`Publisher: Error generating EPUB for ${instance.id}:`, error); if (callback)
               callback();
         });
      } // !isNewer
   }
   .bind(this)); // executeIfPathIsOlder
}
```

ePub code with epub-gen

```
Publisher.prototype.performOutputJobPdf = function(job, callback) {
   // console.log(`Publisher: performOutputJobPdf `+job.instance.id+' as '+job.type);
   var instance = job.instance;
   var language = job.language;
   var languageSuffix = language.toUpperCase();
   var pdfFilename = instance['slug' + languageSuffix] + '.pdf';
   this.base.isFileNewerThanTimestamp(instance['absolutePath' + languageSuffix] + pdfFilename,
                                      instance.lastUpdated, function(isNewer) {
      if (isNewer) {
         callback();
         // no action needed
      } else {
         // the PDF needs to be generated
         console.log('Publisher: generating PDF for ' + job.instance.id);
         var inputHtml = `<html lang="${language}">
                           <head>
                               <title>${instance["title"+languageSuffix]}</title>
                           </head>
                           <body>
                               ${instance["feedHtml"+languageSuffix]}
                           </body>
                       </html>`;
         // the inputHtml is based on feedHtml which uses absolute URL's, since
         // we're processing locally on the filesystem into PDF's, we're removing
         // online (image) links, transforming them in absolute filesystem paths:
         // we also remove the anti cache parameter ?=LASTUPDATED
         inputHtml = this.base.replaceAllInString(this.base.replaceAllInString(inputHtml,
                        'https://' + this.base.rootFolderName + instance['relativePath' + languageSuffix],
                        instance['absolutePath' + languageSuffix]), '?u=' + instance.lastUpdated, '');
         // we can only do the PDF generation if we have the basic html as input,
         // so we make sure that file is written first:
         this.base.makeSureFileIsWritten(instance['absolutePath' + languageSuffix] + 'tmp-pdf-input.html', inputHtml, function() {
            // Then we compose variables used to start a pandoc process to generate a PDF
            // using LateX magic and the basic.html as input:
            const languageSuffix = language.toUpperCase();
            const inputPath = instance['absolutePath' + languageSuffix] + 'tmp-pdf-input.html';
            const outputPath = instance['absolutePath' + languageSuffix] + pdfFilename;
            const templatePath = this.base.rootPath + '/assets/template.latex';
            const title = this.base.escapeLatex(instance['title' + languageSuffix]);
            const subTitle = this.base.escapeLatex(instance['subTitle' + languageSuffix]);
            const author = this.base.escapeLatex(this.authorName);
            const date = instance["dateFormatted" + languageSuffix];
            const heroImagePath = (instance['image' + languageSuffix])
                ? this.base.resultPath + instance['image' + languageSuffix].url.w1000
                : this.base.resultPath + '/global/icon.png';
            // Construct the Pandoc command with the corrected engine
            const command = `pandoc \\
                             -f html \\
                             -o "${outputPath}" \\
                             --pdf-engine=xelatex \\
                             -V title="${title}" \\
                             -V subTitle="${subTitle}" \\
                             -V heroImagePath="${heroImagePath}" \\
                             -V author="${author}" \\
                             -V date="${date}" \\
                             --template="${templatePath}" \\
                             "${inputPath}"`;
            // Execute the command (and pray for some good luck)
            exec(command, (error, stdout, stderr) => {
               if (error) {
                  console.error(`Publisher: Error generating PDF for ${instance.id}: ${error.message}`);
                  callback();
                  // keep continuing on the other jobs
                  return;
               }
               // if (stderr) {
               // console.warn(`Publisher: Pandoc stderr for ${instance.id}: ${stderr}`);
               // }
               console.log(`Publisher: Successfully generated PDF at ${pdfFilename}`);
               // clear tmp input HTML file:
               fs.unlink(inputPath, function(error) {
                  if (error)
                     throw error;
                  console.log('Publisher: Removed tmp HTML file from ' + inputPath);
               });
               callback();
               // tell the job queue we're done by calling back
            });
            // exec pandoc
         }
         .bind(this)); // makeSureFileIsWritten basic.html
      } // !isNewer
   }
   .bind(this)); // executeIfPathIsOlder
}
```

PDF code using Pandoc and LaTeX