

AH Formatter を用いた Markdown-PDF 変換事例紹介

2019 年 5 月

フェリックス・スタイル

はじめに

この文書は、次のような方に向けた事例紹介です。

- Visual Studio Code で、ソフトウェア開発に付随する報告書などの作成も行いたい
- 文書は Markdown で執筆し、PDF で提出したい
- コマンドライン操作のみで PDF ビルドが完了するようにしたい
- GitHub での Markdown 表示のようなスタイリングにしたい
- PDF には表紙や目次、ノンブル（ページ番号）を付けたい
- 目次は、項目タイトルとページ番号、ドキュメント内リンクを自動作成してほしい
- ノンブルは、表紙や目次を除いた本文ページのみに振りたい
- PDF には余計なプロパティを含めたくない

上記の要件を全て満たす PDF を得るために筆者が用いたフローを紹介します。（上記の要件の一部が不要であれば、また違ったフローがあり得るでしょう。）

ここで取り上げる内容は、筆者の環境で手間をかけずに実現するために用いたフローであって、ベストプラクティスではなく、汎用性に欠ける点もありますが、より良い実現方法が共有されていく叩き台になれば幸いです。

目次

• Markdown-PDF 変換.....	1
◦ 動作環境.....	1
◦ md2pdf.....	1
▪ セットアップ.....	1
▪ PDF のビルド.....	2
◦ Markdown 原稿の執筆.....	3
▪ Markdown Preview Github Styling の導入.....	3
▪ 複雑な表の記述.....	3
◦ Markdown 目次の生成.....	4
▪ build:doc-1 Markdown ファイルの連結.....	4
▪ build:doc-2 DocToc による目次生成.....	4
◦ Markdown から HTML へ変換.....	5
▪ build:doc-3 markdown-it による変換.....	5
▪ build:doc-4 Markdown 外部の内容を付与.....	5
▪ build:doc-5 html-inline によるインライン化.....	6
◦ HTML から PDF へ変換.....	7
▪ build:doc-6 AHFCmd による変換.....	7
• 付録 ソースファイル抜粋.....	8
◦ ファイル構成.....	8
◦ package.json.....	9
◦ scripts/mdit.js.....	10
◦ scripts/ejs.js.....	11

Markdown-PDF 変換

Markdown 原稿から、CSS 組版によって表紙や目次、ノンブル等の体裁を備えた PDF にビルドするフローを紹介します。

動作環境

ここで紹介するフローは、以下の環境で動作します。

- Node.js 10+
- AH CSS Formatter V6.6 MR5+

任意で、Visual Studio Code (VS Code)を利用すると、Markdown 編集、プレビュー、PDF へのビルドなどを一つのウィンドウで行えるようになって便利です。

md2pdf

この文書の作成に用いたソースファイル一式「[md2pdf](https://github.com/2SC1815J/md2pdf)」¹⁾をダウンロードして、Markdown 原稿から PDF へのビルドを試すことができます。

セットアップ

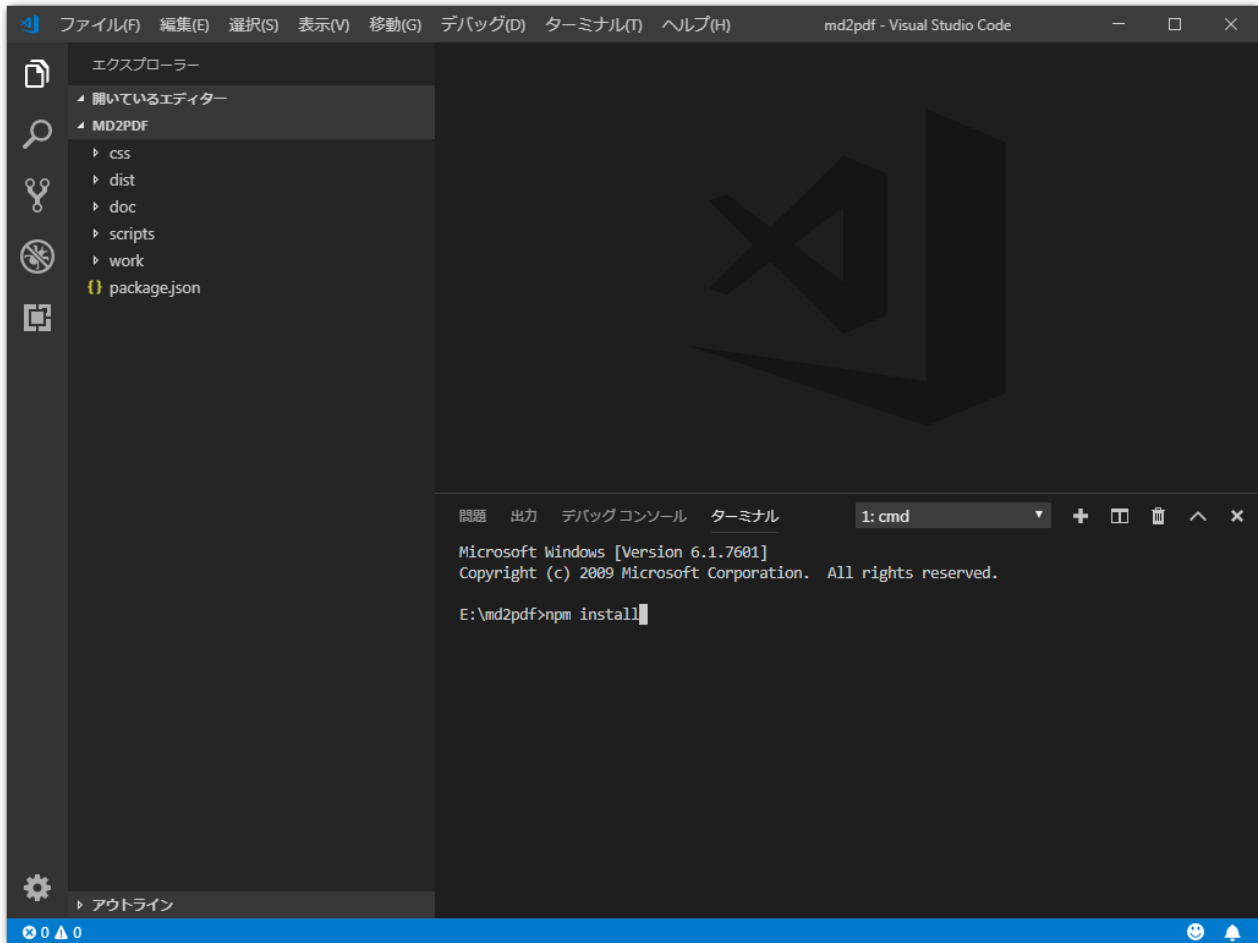
ダウンロードしたソースファイル一式を適当な場所に解凍します。

VS Code を用いる場合、VS Code の「ファイル」メニューから「フォルダーを開く」を選択し、解凍したフォルダを開いてください。続いて、「ターミナル」メニューから「新しいターミナル」を選択します。VS Code を用いない場合は、ターミナル（Windows であればコマンドプロンプトなど）を開き、解凍したフォルダに移動してください。

次に、ターミナルから以下のコマンドを実行してください。Markdown から PDF へビルドするプロセスに必要な Node.js パッケージがインストールされます。

```
npm install
```

1) <https://github.com/2SC1815J/md2pdf>



VS Code のターミナル（右下）

PDF のビルド

ターミナルから次のコマンドを実行することで、doc フォルダ内の Markdown 原稿から HTML と PDF が生成され、dist フォルダに出力されます。²⁾

```
npm run build
```

このビルドプロセスは、package.json ファイルの scripts プロパティに記載された、`build:doc-1` から `build:doc-6` のステップに従って実行されます。

PDF ビルドの前段階としての Markdown 原稿執筆から、PDF ビルドの各ステップまで、順を追って処理内容を見ていきましょう。

2) VS Code を利用している場合は、VS Code の `npm.enableScriptExplorer` 設定を有効にすることで、コマンドを入力しなくてもマウスクリックでビルドを実行できるようになります。

Markdown 原稿の執筆

Markdown Preview Github Styling の導入

VS Code で Markdown 原稿を執筆する場合、VS Code の拡張機能「[Markdown Preview Github Styling](#)」³⁾をインストールすると、Markdown が GitHub 風のスタイリングでプレビュー表示されるようになり、変換結果をイメージしやすくなります。



「Markdown Preview Github Styling」によるプレビュー表示

複雑な表の記述

Markdown では表現力が十分でない部分（セルが結合された表など）は、Markdown 中に HTML で直接記述します。

3) <https://marketplace.visualstudio.com/itemdetails?itemName=bierner.markdown-preview-github-styles>

AH Formatter V6.6 MR4 では、結合されたセルの背景色が正しく設定されないことがありましたが、AH Formatter V6.6 MR5 で修正されました（表 1）。⁴⁾

表 1

見出し 1	見出し 2	
	見出し 2-1	見出し 2-2

Markdown 目次の生成

build:doc-1 Markdown ファイルの連結

目次のひな形（doc/toc.md）を、複数の Markdown 原稿（doc/preface.md, doc/main.md, doc/appendix.md）と合わせて、単一の Markdown ファイルにします。

```
npx minicat doc/preface.md doc/toc.md doc/main.md doc/appendix.md > work/all.md
```

このステップによって、仮 Markdown ファイル（work/all.md）が生成されます。

build:doc-2 Doctoc による目次生成

Node.js パッケージ「[Doctoc](https://github.com/thlorenz/doctoc)」⁵⁾を用いて、目次を生成します。Markdown に記述された見出し要素が抽出され、目次が自動的に作成されます。

```
npx doctoc --notitle --maxlevel 3 work/all.md
```

このステップによって、仮 Markdown ファイル（work/all.md）の目次が更新されます。

⁴⁾ AH Formatter V6.6 MR4 までの場合、明示的にクラス指定を行うことで対処します。

⁵⁾ <https://github.com/thlorenz/doctoc>

Markdown から HTML へ変換

build:doc-3 markdown-it による変換

Node.js パッケージ「[markdown-it](#)」⁶⁾を用いて、Markdown から HTML へ変換します。

```
node scripts/mdit.js work/all.md work/all_md.html
```

Markdown の見出し要素を日本語で記述しており、目次部分の href に URI エンコードされた日本語文字列が設定されている場合、AH Formatter V6.6 MR4 では、CSS 組版によるページ番号表示 (target-counter(attr(href))の処理) が正しく動作しない問題がありましたが、AH Formatter V6.6 MR5 で修正されました。⁷⁾

このステップによって、部分的な HTML ファイル (work/all_md.html) が生成されます。

build:doc-4 Markdown 外部の内容を付与

markdown-it が生成する HTML には、<html>や<head>、<body>、CSS ファイルへのリンクなどは含まれていません。これらを記述したテンプレート HTML ファイル (doc/template.html) を別途用意し、markdown-it によって生成された HTML がその中に含まれるようにします。今回は表紙の内容もここに記述しました。

```
node scripts/ejs.js doc/template.html work/all.html
```

AH Formatter V6.6 では、HTML ファイルの meta タグに次の記載を含めることで、この HTML から変換された PDF が開かれる際、ページ全体がウィンドウに収まるズーム状態とすることができます。

```
<meta name="openaction" content="#view=fit">
```

6) <https://github.com/markdown-it/markdown-it>

7) AH Formatter V6.6 MR4 までの場合、Markdown の見出し要素からアンカーを作成する Node.js パッケージ「[anchor-markdown-header](#)」に変更を加えて対処する方法があります。

また、AH Formatter V6.6 MR5 からは、次の記載を含めることで、生成される PDF の文書情報に含まれる作成日 (/CreationDate) と更新日 (/ModDate) を指定の値に設定することができます。⁸⁾

```
<meta name="creationdate" content="20190501T090000+09">
<meta name="modifydate" content="20190501T090000+09">
```

GitHub での Markdown 表示のようなスタイリングを実現するには、CSS ファイル「[github-markdown-css](#)」⁹⁾が利用できます (css/github-markdown.css)。

CSS 組版用の CSS ファイル (css/custom.css) を、「[CSS ページ組版入門](#)」¹⁰⁾などを参考にして記述し、テンプレート HTML ファイルから読み込まれるようにしておきます。

このステップによって、仮 HTML ファイル (work/all.html) が生成されます。

CSS 組版用の CSS ファイルを調整する場合は、この HTML ファイルを AH Formatter のグラフィカルユーザインターフェイスで読み込み、組版結果を確認しながら行うと良いでしょう。

build:doc-5 html-inline によるインライン化

Node.js パッケージ「[html-inline](#)」¹¹⁾を用いて、HTML ファイルから参照されている CSS ファイルや画像ファイルを HTML ファイル内にインライン化 (埋め込み) します。¹²⁾

```
npx html-inline work/all.html -b doc -o dist/all.html
```

このステップによって、必要な情報が全て含まれた HTML ファイル (dist/all.html) が生成されます。

8) AH Formatter V6.6 MR4 までの場合、「[Coherent PDF Command Line Tools \(cpdf\)](#)」を利用して設定する方法があります。

9) <https://github.com/sindresorhus/github-markdown-css>

10) https://www.antenna.co.jp/AHF/ahf_publication/index.html#CSSPrint

11) <https://github.com/substack/html-inline>

12) AH Formatter V6.6 MR5 までの場合、生成される PDF のプロパティ「ベース URL」に生成元 HTML ファイルの場所が表示され、同表示を空にするには、HTML ファイルから参照されている外部ファイルをインライン化した上で、PDF 出力時にコマンドラインパラメータ指定 (-base "") を行う必要がありました。AH Formatter V6.6 MR6 からは、これらの処理は不要となっています。

HTML から PDF へ変換

build:doc-6 AHFCmd による変換

AH Formatter のコマンドラインインターフェイス AHFCmd を用いて、必要な情報が全て含まれた HTML ファイルを CSS 組版し、PDF ファイルとして出力します。

```
AHFCmd -d dist/all.html -p @PDF -pdfver 1.5 -base " " -x 4 -pgbar -o dist/all.pdf
```

このステップによって、最終的な PDF ファイル（dist/all.pdf）が生成されます。

付録 ソースファイル抜粋

この文書の作成に用いたソースファイル一式のファイル構成と、主要なファイルの内容を挙げます。

ファイル構成

```
md2pdf
├── package.json
├── css
│   ├── custom.css
│   └── github-markdown.css
├── dist
│   ├── all.html
│   └── all.pdf
├── doc
│   ├── appendix.md
│   ├── fig001.png
│   ├── fig002.png
│   ├── main.md
│   ├── preface.md
│   ├── template.html
│   └── toc.md
├── node_modules
├── scripts
│   ├── ejs.js
│   └── mdit.js
└── work
```

package.json

```
{
  "name": "md2pdf",
  "version": "1.0.0",
  "description": "Convert Markdown documents to PDF",
  "scripts": {
    "build:doc-1": "npx minicat doc/preface.md doc/toc.md doc/main.md doc/
appendix.md > work/all.md",
    "build:doc-2": "npx doctoc --notitle --maxlevel 3 work/all.md",
    "build:doc-3": "node scripts/mdit.js work/all.md work/all_md.html",
    "build:doc-4": "node scripts/ejs.js doc/template.html work/all.html",
    "build:doc-5": "npx html-inline work/all.html -b doc -o dist/all.html",
    "build:doc-6": "AHFCmd -d dist/all.html -p @PDF -pdfver 1.5 -base \" \" -x
4 -pgbar -o dist/all.pdf",
    "build": "npm run build:doc-1 && npm run build:doc-2 && npm run build:doc-3
&& npm run build:doc-4 && npm run build:doc-5 && npm run build:doc-6"
  },
  "keywords": [
    "Markdown",
    "PDF"
  ],
  "author": "2SC1815J",
  "license": "MIT",
  "devDependencies": {
    "anchor-markdown-header": "^0.5.7",
    "doctoc": "^1.4.0",
    "ejs": "^2.6.1",
    "eslint": "^5.16.0",
    "html-inline": "^1.2.0",
    "htmltidy2": "^0.3.0",
    "markdown-it": "^8.4.2",
    "markdown-it-implicit-figures": "^0.9.0",
    "markdown-it-named-headers": "0.0.4",
    "minicat": "^1.0.0"
  }
}
```

scripts/mdit.js

```
/*
 * md2html
 * Copyright 2019 2SC1815J, MIT license
 */
'use strict';
if (process.argv.length < 4) {
  console.error('Usage: node mdit.js input.md output.html');
  process.exit(1);
}

const header_instances = {};
const anchor = require('anchor-markdown-header');
const mdit = require('markdown-it')(
  {
    html: true
  })
  .use(require('markdown-it-named-headers'), {
    slugify: function(header) {
      if (header_instances[header] !== void 0) {
        header_instances[header]++;
      } else {
        header_instances[header] = 0;
      }
      const match = anchor(header, 'github.com',
header_instances[header]).match(/\((#(.*?)\)$/);
      return match ? decodeURI(match[1]) : header;
    }
  })
  .use(require('markdown-it-implicit-figures'), {
    figcaption: true
  });

const { promisify } = require('util');
const fs = require('fs');

(async () => {
  const md = await promisify(fs.readFile)(process.argv[2], 'utf8');
  const html = mdit.render(md);
  await promisify(fs.writeFile)(process.argv[3], html, 'utf8');
})();
  .then(() => {
```

```
        console.log('Done.');
```

```
    })
    .catch((err) => {
        console.error(err);
        process.exit(1);
    });
```

scripts/ejs.js

```
/*
 * md2html
 * Copyright 2019 2SC1815J, MIT license
 */
'use strict';
if (process.argv.length < 4) {
    console.error('Usage: node ej.js template.html output.html');
    process.exit(1);
}

const { promisify } = require('util');
const ej = require('ejs');
const tidy = require('htmltidy2');
const fs = require('fs');

(async () => {
    const text = await promisify(ej.renderFile)(process.argv[2]);
    const options = {
        doctype: 'html5',
        indent: 'auto',
        wrap: 0,
        tidyMark: false,
        quoteAmpersand: false,
        hideComments: true,
        dropEmptyElements: false,
        newline: 'LF'
    };
    const tidied = await promisify(tidy.tidy)(text, options);
    await promisify(fs.writeFile)(process.argv[3], tidied, 'utf8');
})();

.then(() => {
    console.log('Done.');
```

```
.catch((err) => {  
  console.error(err);  
  process.exit(1);  
});
```

