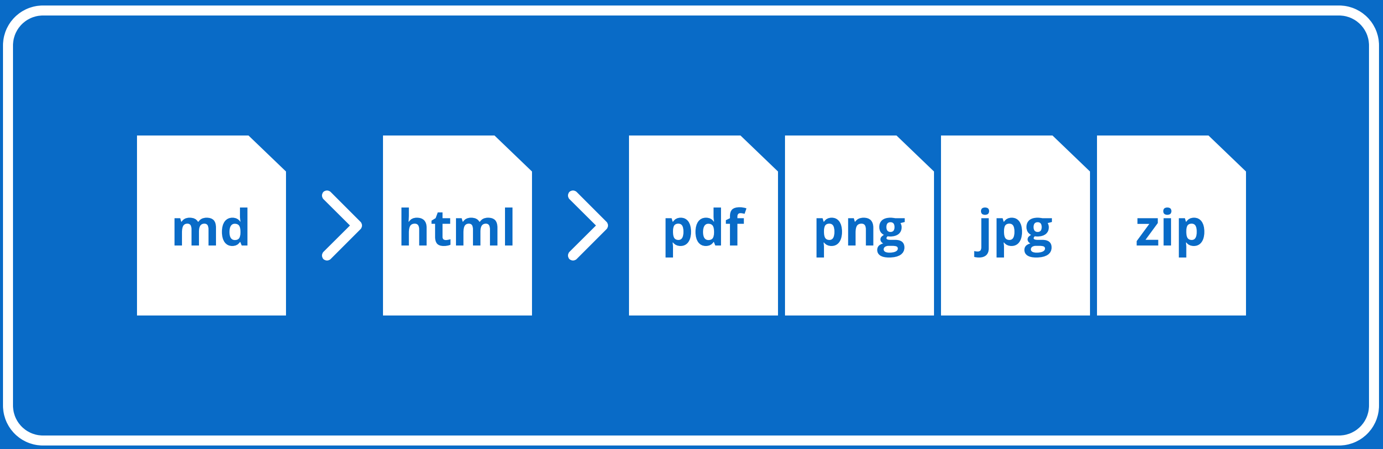
GIFFYGLYPH'S PROJECT PUBLISHER

Instructions & Tutorial



Contents

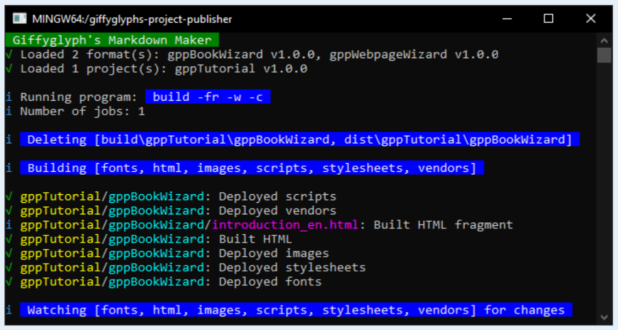
Giffyglyph's Project Publisher

What is Publisher?

Giffyglyph's Project Publisher (aka Publisher) is a fully-featured content pipeline and CLI tool that turns markdown documents into HTML, PDFs, PNGs, and more in just five simple steps:

1. **Start** a new Publisher project and choose one or more rendering formats—are you making a book, post, webpage, or website?
2. **Write** a document in plain-text markdown.
3. **Enrich** it with stylesheets, scripts, and translations.
4. **Build** HTML files from your document.
5. **Export** your HTML as PDFs, PNGs, JPGs, or ZIPs.

When you want to write, refine, and release markdown documents, Publisher has you covered.



```
MINGW64:/giffyglyphs-project-publisher
 Giffyglyph's Markdown Maker
√ Loaded 2 format(s): gppBookWizard v1.0.0, gppWebpageWizard v1.0.0
√ Loaded 1 project(s): gppTutorial v1.0.0

i Running program:  build -fr -w -c
i Number of jobs: 1

i  Deleting [build\gppTutorial\gppBookWizard, dist\gppTutorial\gppBookWizard]

i  Building [fonts, html, images, scripts, stylesheets, vendors]

√ gppTutorial/gppBookWizard: Deployed scripts
√ gppTutorial/gppBookWizard: Deployed vendors
i gppTutorial/gppBookWizard/introduction_en.html: Built HTML fragment
√ gppTutorial/gppBookWizard: Built HTML
√ gppTutorial/gppBookWizard: Deployed images
√ gppTutorial/gppBookWizard: Deployed stylesheets
√ gppTutorial/gppBookWizard: Deployed fonts

i  Watching [fonts, html, images, scripts, stylesheets, vendors] for changes
```

The *Project Publisher* CLI window

Formats & Projects

Publisher relies on two categories of content:

1. **Formats:** "What *type* of content am I trying to publish, and how should it behave?" Formats are reusable "skins" that define the basic shape, functionality, and appearance of your projects.
2. **Projects:** "What *actual* content I am trying to publish?" Projects are where you put your text and anything else you might want to override the underlying format with (stylesheets, fonts, images, scripts).

By dividing your content into formats and projects, you can build your own library of reusable publishing components.

Write in Markdown

Create documents with markdown—a flexible, easy-to-learn/easy-to-master, plain text markup language.

Need to use some specific HTML in your document as well? No problem—markdown supports HTML too!

Build HTML

Turn your markdown into fully-featured HTML pages with *rendering formats*. These formats come in one of four categories:

- **Book:** Render markdown as standard-sized *pages* that are bound into a *book*.
- **Post:** Render markdown as *posts* that can be of varying size, shape, style, etc.
- **Webpage:** Render markdown as a single webpage.
- **Website:** Render markdown as a multi-page website complete with navigation.

You can assign multiple formats to each project and build them all *simultaneously* with one command.

Export Artifacts

Export and share your rendered HTML files as PDFs, PNGs, JPGs, and ZIPs.

Formats can be configured to export files with varying options, and you can easily extend a format with new export options if-and-when the need arises.

Valiant wants to write a short 10-page campaign primer for a D&D game he's running. He creates a new project, writes the text in simple markdown, renders it using a premade "book" format, and exports it as a PDF for his players to read.

Some time later, Valiant decides to release the primer online as a website for others to read. He renders it using a "website" format, exports it as a ZIP, and uploads the files to his webhost.

Style with SASS

Want to change how your documents look for a more *personal* touch? It's easy with SASS stylesheets!

Make some small tweaks (swap some colors, change fonts, add a new theme, etc.) or go all-in and completely overhaul the layout of your content.

Clanda is working on a new TTRPG and wants to create some *posts* for her social media. She creates a new project with a "post" format and adds *custom stylesheets*, changing the color scheme and font choices to better match the theming of her TTRPG.

Language Translations

Translate your content into other languages with message keys and translation files, and render translated copies of your HTML with no additional effort.

Scripts and Scripting

It's easy to transform, extend, and automate your markdown documents—just use javascript!

Add third-party libraries (jQuery, moment.js, bootstrap, etc.) or create your own scripts to use custom features and functionality. With *JSON renderers*, you can even write JSON objects directly into your markdown.

Fragments & Collections

Use Publisher to break up large documents into small, easy-to-manage *fragments*. You can bind these fragments together in any fashion with *collections*.

Extensions

Formats allow you to customise almost every part of Publisher—change how HTML is rendered, create new export types, override how markdown is parsed, etc.

Create your own publishing formats to suit your own bespoke projects, or import them from the community.

Why use Publisher?

If any of the following sound familiar to you, then *Publisher* might be a good fit for your projects:

- I want to write in plain text.
- I want to change the appearance of my content.
- I want to automate parts of my text.
- I want full control of my project code.
- I want to break up a monolithic document into smaller fragments for easier management.
- I want to publish my content in multiple formats.
- I want to translate my content into other languages.
- I want to export and share my work easily with others.
- I want to import new formats from the community and use them in my own projects.

Use Publisher when you want total control over your content and how it's produced.

Published Examples

This handbook was written for, rendered in, and exported by *Giffyglyph's Project Publisher*. Other examples of Publisher-authored projects include:

- **Darker Dungeons:** A *D&D 5e* book (PDF, website).
- **Monster Maker:** A *D&D 5e* book (PDF, website).
- **Class Compendium:** A *D&D 5e* book (PDF, website).
- **Card Codex:** A *D&D 5e* supplement (PDF).
- **Quick Quest:** A rules-lite RPG system (PDF).
- **Giffyglyph's Project Posts:** A visual log of project status updates (PNGs).

License

Publisher (and the underlying *Markdown Maker* engine) is released under a GNU GPL v3.0 license. In summary:

- You may copy, modify and distribute this software.
- You must include the license and copyright notice with each and every distribution.
- Any modifications of this code base MUST be distributed with the same license—GPLv3.
- This license does *not* cover your own formats, projects, or artifacts (PDFs, PNGs, etc.)—anything you create using this software remains your own work.

Support

If you'd like to see more from this tool—or would like to commission a custom format for your project—consider becoming a patron or making a one-off donation.

Become a patron

Buy me a coffee

Contact Me

Special Thanks

This tool was made possible with the support of:

Adam Mote, Ady Veisz Dragia, AlDragonus, Alexander von Bose, Anthony Campla, Arthur Bauer, Barkston, Berke Canatar, Birb Marrows, damon1245, Dario Hajic, FC Wesel, Faolan Twinbear, Hoots Kenku, Michael C. Miller, Necrotic Fawn, Strider

...and 393 other patrons.

Getting Started

Installation & First Run

To install and run *Giffyglyph's Project Publisher* software on your machine, follow these six simple steps:

1 Install Node.js

First, you must download and install node.js v14.17.6. Once installed, run the following terminal commands to verify that both node and npm are running correctly:

```
node -v
npm -v
```

2 Install Publisher

Next, download the latest release of Publisher (available at github.com/giffyglyph/giffyglyphs-project-publisher) and extract the files to a folder on your local machine—for example, "C:/giffyglyphs-project-publisher".

Example Content: Publisher comes complete with one example project (gppTutorial) and one example format (gppBookWizard)—find these in the "src" folder.

3 Install Modules

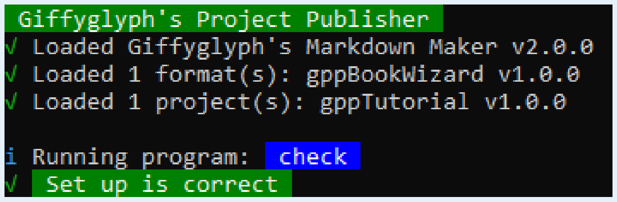
Now, open up a terminal window and navigate to your *Publisher* folder. Run the following command to automatically install all prerequisite node modules:

```
npm install
```

4 Run your First Command

Once all modules are downloaded, run the following command to verify that Publisher is working correctly:

```
node publisher.js check
```



```
 Giffyglyph's Project Publisher
√ Loaded Giffyglyph's Markdown Maker v2.0.0
√ Loaded 1 format(s): gppBookWizard v1.0.0
√ Loaded 1 project(s): gppTutorial v1.0.0

i Running program:  check
√  Set up is correct
```

If you see "Set up is correct", you're ready to publish!

Publisher Configuration

Once installed, open up the **publisher.js** file to check your configuration. There are five main points of interest:

1. **Title:** The title to be shown in the command window (i.e. "Giffyglyph's Project Publisher").
2. **Output/Build:** This specifies which folder your build HTML files should be saved to.
3. **Output/Export:** This specifies which folder your exported artifacts should be saved to.
4. **Formats:** This is a list of all formats the software should try to load.
5. **Projects:** This is a list of all projects the software should try to load.

5 Build some HTML

You're ready to start building some HTML files! Open up your terminal window, navigate to your Publisher folder, and run the following command:

```
node publisher.js build -p gppTutorial -f
gppBookWizard
```

Publisher will read the collection files in the example project (i.e. "src/gppTutorial/collections"), gather the markdown files from the *fragments* folder, create the HTML, and build/deploy any additional files (stylesheets, fonts, images, etc.).

Once the program is complete, you can find your built files in "build/gppTutorial/gppBookWizard/html".

6 Export a PDF

After you build some HTML from your markdown, you can *export* it! Use the following command to create a PDF from your new HTML file:

```
node publisher.js export -ex pdf -p
gppTutorial -f gppBookWizard
```

Publisher will gather files from the build folder (i.e. "build/gppTutorial/gppBookWizard") to create a PDF.

Once the program is complete, you can find your new PDF in "export/gppTutorial/gppBookWizard/pdf".

Running Programs

Publisher is operated by *terminal commands* which run *programs*. There are five core programs to use—build, export, clean, watch, and check:

Build

The *Build* program turns your raw markdown into HTML, creates CSS stylesheets from your SASS, and deploys any additional content required by your artifacts—scripts, fonts, images, etc.

Fragments and Collections: By default, *Build* looks at your *collections* to decide a) *which* markdown files it should use and b) *how* to package them. But if you want to build individual fragments instead (i.e. for testing or development), use the "-fr" option (listed below).

```
node publisher.js build
```

Option	Description
-p, --projects <name...>	One or more loaded projects.
-f, --formats <name...>	One or more loaded formats.
-l, --languages <code...>	One or more languages.
-t, --tasks <task...>	Run a selection of build sub-tasks (html, stylesheets, scripts, fonts, images, or vendors).
-fi, --files <name...>	Build specific files.
-fr, --fragments	Build as fragments only.
-c, --clean	Run *Clean* before building.
-w, --watch	Run *Watch* after building.
-d, --debug	Show debug information.
-di, --discrete	Use no colors in the log.
-h, --help	List all program options.

Export

The *Export* program turns your built HTML into artifacts —PDF, PNGs, JPGs, or ZIPs.

Build First: Export uses files from the build folder, so make sure to run a fresh *Build* before you try exporting.

```
node publisher.js export
```

Option	Description
-p, --projects <name...>	One or more loaded projects.
-f, --formats <name...>	One or more loaded formats.
-fi, --files <name...>	Export specific files.
-ex, --export	Type of export (pdf, png, jpg, or zip).
-pg, --pages	Export a specific page range.
-d, --debug	Show debug information.
-di, --discrete	Use no colors in the log.
-h, --help	List all program options.

Parallel Activity

Some command options allow you to specify a space-separated list of names/codes/tasks/etc. Publisher will run multiple jobs *in parallel*, saving you time when running bulk actions.

Defaulting to All: Some options (projects, formats, tasks) default to "all" if unspecified.

Clean

The *Clean* program deletes build and export folders. This program is commonly used to clear out old artifacts before a build.

```
node publisher.js clean
```

Option	Description
-p, --projects <name...>	One or more loaded projects.
-f, --formats <name...>	One or more loaded formats.
-d, --debug	Show debug information.
-di, --discrete	Use no colors in the log.
-h, --help	List all program options.

Watch

The *Watch* program keeps an eye on source files and folders—when any change is detected, a *Build* is then automatically triggered. This can be useful when

```
node publisher.js watch
```

Option	Description
-p, --projects <name...>	One or more loaded projects.
-f, --formats <name...>	One or more loaded formats.
-t, --tasks <task...>	Trigger a selection of build sub-tasks (html, stylesheets, scripts, fonts, images, or vendors).
-fi, --files <name...>	Watch specific files.
-fr, --fragments	Build as fragments only.
-d, --debug	Show debug information.
-di, --discrete	Use no colors in the log.
-h, --help	List all program options.

Check

The *Check* program confirms that your Publisher configuration is working, verifying the core configuration and any specified formats/projects.

```
node publisher.js check
```

Markdown

Writing Markdown

Publisher documents are written in markdown—a simple and easy-to-use markup language. To learn how to use markdown, visit the Markdown Guide.

- **Basic Syntax:** markdownguide.org/basic-syntax/.
- **Cheat Sheet:** markdownguide.org/cheat-sheet/.

Publisher Extensions

Publisher includes a number of extensions to the markdown language to help you group, tag, and layout your content more easily. Use these to enrich your document with format-specific elements.

Extending Extensions: These markdown extensions can be easily styled with stylesheets or *entirely* overidden by formats for extra customisation.

Headings

You can add HTML attributes easily to your headings—use these to add custom classes, ids, or data attributes.

```
# Heading 1 { "id": "x", "class": "y" }
## Heading 2 { "data-toc-ignore": true }
```

Content Block

Publisher formats (such as "book" and "post") require you to divide your content into sections—i.e. *pages* and *posts*. Use *content blocks* to do this.

```
\contentBegin { "id": "x", "class": "y" }
Write your content here.
\contentEnd
```

Panel

Use a *panel* when you want to add some highlighted aside text to your document.

Title: Specify a "title" to automatically insert a heading into the panel.

```
/panelBegin { "title": "A title" }
Write your panel body here.
/panelEnd
```

HTML Attributes

HTML attributes (id, class, data attributes, etc.) are an important means of customising your content.

You can add attributes easily to these extensions with a simple, JSON-compatible string.

Column Break

Column breaks allow you to push all subsequent content into a new column.

```
\columnbreak
```

Table Block

Use a *table block* when you need to insert a table.

Title: Specify a "title" to automatically insert a heading into the table.

```
/tableBegin { "title": "A title" }
|column1|column2|
|-------|-------|
|data A1|data B2|
/tableEnd
```

Example

Use an *example block* when you want to add some secondary text to your document—such as an example, tutorial, or clarification.

```
/exampleBegin { "id": "x", "class": "y" }
This is an example of an example block.
/exampleEnd
```

Figure

Use a *figure block* when you want to insert an image.

Caption: Specify a "caption" to automatically insert a figcaption into the figure.

```
/figureBegin { "caption": "A caption" }
[image url]
/figureEnd
```

Projects

What is a Project?

A *project* is a bundle of content you want to publish—this is where you put the raw material (markdown, images, project-specific stylesheets, etc) for your artifacts.

Projects can be assigned any number of publishing formats (i.e. "skins") to transform their appearance and behaviour—in this way, you can reuse the *same* project files to create wildly different artifacts.

Folder Structure

Projects are stored in "src/projects". When you add files to your project, store them in the following folders:

- **Collections:** JSON files that describe how to bind fragments together.
- **Fonts:** Distributable font files.
- **Fragments:** Markdown files that are rendered into distributable HTML.
- **Modules:** Non-distributed javascript that may be used by the project configuration.
- **Scripts:** Distributable javascript files.
- **Stylesheets:** SASS stylesheets that are compiled into distributable CSS.
- **Translations:** Translated message keys.
- **Vendors:** Distributable third-party libraries.

Format-specific Content

By default, all of your project content is available to every format. However, you want some content to be used only by a *specific* format—for example, using one image for a "book" format and one for a "website" format.

When you want to restrict access to some content, put it in a format subfolder ("_<formatName>"):

Chansi is creating a new project (chaRecipes) that uses multiple formats. One file (file3.md) is only to be used with the "gppBookWizard" format, so she puts it into a format subfolder.

- chaRecipes/src
  - fragments
    - file1.md
    - file2.md
    - _gppBookWizard
      - file1.md
      - file3.md

File Priority

When Publisher looks for *content* (e.g. a font, an image, a stylesheet, etc.), it searches in three areas of descending priority:

1. project/format
2. project
3. format

Higher priority files will always overwrite lower priority files when running *Build* and *Export*.

Adding a Project

To add an existing project to your Publisher:

- **Copy** the project into your "src/projects" folder.
- **Update** your publisher.js config file to reference the new project's .js file.
- **Confirm** that your project loads correctly by running the *Check* program.

Some projects may be configured to require additional formats, so make sure to install these if missing.

Viridian is working with Chansi on the "chaRecipes" project. He downloads the project files, extracts them to his src/projects folder, and updates his publisher.js config.

Creating a Project

To create your own Publisher project, follow these three starting steps:

1 Choose a Name

You first step is to choose a project name. To help share your project with others, use this naming convention:

- **Creator Initials:** Pick 3 lowercase letters to serve as your initials.
- **Project Codename:** A codename that represents the content of your project.

Project Config

Parameter	Type	Description
version	string	The current version of this project.
author	string	Who made this?
description	string	What is this project about?
processHtml	function(dom)	Must return a dom element.
format	object	What formats does this project require?

2 Create a Folder & Config

Next, create a new folder and configuration file for your project. This is done in the following location:

```
src/project/<projectName>/<projectName>.js
```

Your initial config file should resemble:

```
const project = new Project({
    version: "1.0.0",
    formats: []
});
```

3 Add some Formats

A project needs some formats—otherwise, Publisher won't know how to build or export your content. To get started, add a format to your project by specifying:

- **Name:** What's the name of the target format? If no loaded format has this exact name, Publisher will post an error.
- **Version:** What *version* of the target format is required? If the loaded format has a different version, Publisher will post an error.
- **Languages:** What languages do you support building and exporting in with this format?

```
formats: [
    {
        name: "gppBookWizard",
        version: "1.0.0",
        languages: [ "en" ]
    }
]
```

4 Update Publisher

Now add your project to the **publisher.js** config file and run the *Check* program to ensure that everything is working as expected.

If the set up is correct, your project is ready for some new content!

5 Add some Fragments

Fragments contain the markdown text that is the core of your project. Each fragment represents a piece of content, such as a single chapter or post—use these to divide your project into manageable chunks.

Create two new markdown fragments (.md files) in your project's "fragments" folder and add some suitable text. Run the *Build* command with the *fragments* option to test your content:

```
node publisher.js build -p <projectName> -f
```

Open your *build* folder to verify that your fragments have been built correctly.

6 Add a Collection

Collections are how you bind disparate fragments into larger, complete artifacts. A basic collection has an output filename and a list of contents:

```
{
    "filename": "collection01",
    "contents": [
        "file01",
        "file02"
    ]
}
```

Create a new collection (.json file) in your project's "collections" folder and add some fragments to it. Run the *Build* command to test your new collection:

```
node publisher.js build -p <projectName>
```

7 Enrich and Export

Now that you have your basic project up-and-running, you can start to *enrich* it! Try experimenting with markdown blocks, or creating stylesheets to change the appearance, or adding some language translations.

Once you've finished enriching your text, try rebuilding and then exporting it!

Formats

What is a Format?

A *format* is a reusable build-once/apply-many "skin" that defines the basic shape, functionality, and appearance of your markdown. Formats can extend—or override—core parts of the Publisher engine, granting you full control over how your content is processed.

There are four basic types of format: books, posts, webpages, and websites.

Book

Your markdown is divided into "pages" of the same size and shape—use *content blocks* and *column breaks* to lay out your content across these pages. Book formats are a good choice when you want to create PDFs.

Post

Your markdown is divided into "posts" of varying size—use *content blocks* to lay out your content across these posts. Post formats are a good choice when you want to create images.

Webpage

Your markdown is combined into a single webpage. Webpage formats are a good choice when you want to upload some *simple* content to the web.

Website

Your markdown is rendered across several pages with shared navigation. Website formats are a good choice when you want to upload *extensive* content to the web.

Folder Structure

Formats are stored in Publisher's "src/formats" folder. A format may contain these folders:

- **Fonts:** Distributable font files.
- **Images:** Distributable image files.
- **Modules:** Non-distributed javascript that helps to define how the format runs programs.
- **Scripts:** Distributable javascript files.
- **Stylesheets:** SASS stylesheets that are compiled into distributable CSS.
- **Vendors:** Distributable third-party libraries.

Adding a Format

To add an existing format to your Publisher:

1. **Copy** the format into your "src/formats" folder.
2. **Update** your publisher.js config file to reference the new format's .js file.
3. **Confirm** that your format loads correctly by running the *Check* program.

Viridian has downloaded a format that he wants to use for his *webpage* project. He adds the format to his src/formats folder, updates his publisher.js config, and runs *Check* to confirm that every works.

Creating a Format

To create your own Publisher format, follow these three starting steps:

1 Choose a Type & Name

Once you have decided the type of format you are making (book, post, webpage, or website?), you must then choose a name. To help share your format easily with others, use the following naming convention:

- **Creator Initials:** Pick 3 lowercase letters to serve as your initials.
- **Format Type:** The type of your format (book, post, webpage, or website).
- **Format Codename:** A codename that represents the theme or purpose of your format.

Valiant wants to name his D&D campaign primer book format—he chooses valBookPrimer.

Clanda also chooses a name for her TTRPG post format, opting for clnPostWildstorm.

2 Create a Folder & Config

Next, you must create a new folder and configuration file for your format. This is done in the following location:

```
src/formats/<formatName>/<formatName>.js
```

Once you've created your config file, it's time to start configuring your format! To help you set things up, the *Format Config* table below lists every major parameter accepted by the config file.

Export
If you want your format to be able to export a particular type of artifact (e.g. PDF, PNG, JPG, ZIP), you must add it to the *export* section of your config.
You can also specify any additional options for each export type here—page size, resolution, quality, etc.

Override
When you want to override a part of the core Publisher engine, create an override function. All possible overrides—including their required parameters and return types—are listed in the table below.

Markdown
When you want to change how markdown is rendered in your format (content blocks, examples, tables, etc.), create a markdown renderer.

JSON
JSON renderers allow you to write JSON into your markdown and transform it into content blocks. This can be a *very* powerful automation tool, especially when working with data-heavy content.

3 Add it to Publisher

Finally, add your format to the **publisher.js** config file. Run the *Check* program to ensure that everything is working as expected—if the set up is correct, you can start adding your format to your projects!

Format Config

Parameter	Type	Description
version	string	The current version of your format.
author	string	Who made this?
description	string	What does this format do?
publisher	string	What version of Publisher is required?
export.pdf	object	Configuration options for PDFs.
export.png	object	Configuration options for PNGs.
export.jpg	object	Configuration options for JPGs.
export.zip	object	Configuration options for ZIPs.
processHtml	function(dom)	Must return a dom element.
override.buildHtml	function(jobs)	Must return a Promise.
override.buildHtmlFragments	function(job, language)	Must return a Promise.
override.buildHtmlCollections	function(job, language)	Must return a Promise.
override.buildScripts	function(jobs)	Must return a Promise.
override.buildStylesheets	function(jobs)	Must return a Promise.
override.buildImages	function(jobs)	Must return a Promise.
override.buildFonts	function(jobs)	Must return a Promise.
override.buildVendors	function(jobs)	Must return a Promise.
override.exportPdf	function(job, file, options)	Must return a Promise.
override.exportPngs	function(job, file, options)	Must return a Promise.
override.exportJpgs	function(job, file, options)	Must return a Promise.
override.exportZip	function(job)	Must return a Promise.
override.renderHtmlFragmentWrapper	function(job, filename, html)	Must return a string.
override.renderHtmlCollectionWrapper	function(job, filename, html)	Must return a string.
override.saveHtmlFragment	function(job, language, stream)	—
override.saveHtmlCollection	function(job, language, stream)	—
override.validateCollectionJson	function(json)	Must return an object.
override.renderCollectionJson	function(job, json)	Must return a string.
markdown.table	function(job, filename, token, tags)	Must return a string.
markdown.colbreak	function(job, filename, token, tags)	Must return a string.
markdown.layout	function(job, filename, token, tags)	Must return a string.
markdown.page	function(job, filename, token, tags)	Must return a string.
markdown.example	function(job, filename, token, tags)	Must return a string.
markdown.panel	function(job, filename, token, tags)	Must return a string.
markdown.heading	function(level, title, tags)	Must return a string.
json	object	Define any json processors.