GitBook Developers

GitBook

Published with GitBook

Table of Contents

GitBook Developers

This documentation is intended to get you up-and-running with the GitBook APIs. We'll cover everything you need to know, from authentication, to manipulating results, to combining results with other services.

Feel free to fork, clone, and improve this documentation on GitHub.

HTTP API

This describes the resources that make up the official GitBook API. If you have any problems or requests please contact support.

Libraries

Some official libraries are available to easily get you started:

Name	Language
node-gitbook-api	Node.js
go-gitbook-api	Go (Golang)

Schema

All API access is over HTTPS, and accessed through `https://api.gitbook.com/` . All data is sent and received as **JSON**.

```
$ curl -i https://api.gitbook.com/author/gitbookio

HTTP/1.1 200 OK
Server: Cowboy
Connection: keep-alive
Content-Type: application/json; charset=utf-8
Content-Length: 275
Etag: W/"113-25d22777"
Date: Thu, 11 Jun 2015 14:49:26 GMT
Via: 1.1 vegur

{ ... }
```

Authentication

There is currently only one way to authenticate through GitBook API: **Basic Auth**.

Requests that require authentication will return `404 Not Found`, instead of `403 Forbidden`, in some places. This is to prevent the accidental leakage of private books to unauthorized users.

```
$ curl -u "username:token" https://api.gitbook.com/books/
```

Error Format

Error are returned as JSON:

```
{
    "error": "Not found",
    "code": 404
}
```

Pagination

Requests that return multiple items will be paginated to 30 items by default.

You can specify further pages with the `?skip` parameter. For some resources, you can also set a custom page size up to 100 with the `?limit`.

Paginated results will be returned with information about the page context:

```
{
    list: [],
    skip: 0,
    limit: 100,
    total: 0
}
```

Books

List your books

List books for the authenticated user. This includes books from organizations the user can access.

```
GET /books/
```

You can include only books created by the authenticated user:

```
GET /books/author
```

List all public books

This provides a dump of every public repository, in the order that they were created.

```
GET /books/all
```

Get details about a book

Details not included in list of books are included when querying a specific book:

```
GET /book/:username/:id
```

Authors

Get details about an author

AN API method is available to get information about an user or an organization:

```
GET /author/:username
```

Rousseau

Rousseau provides an API to easily proofready and spellcheck texts.

The proofreading API is built on top of the open source utility Rousseau.

The spellchecker is using the Hunspell dictionaries.

The API is accessible at `https://rousseau.gitbook.com`.

Proofread a text

```
POST https://rousseau.gitbook.com/document

{ "document": "Hello World" }
```

Spellcheck a list of words

```
POST https://rousseau.gitbook.com/words

{ "words": ["Hello", "World"] }
```

OPDS

The Open Publication Distribution System (**OPDS**) is an application of the Atom Syndication Format intended to enable content creators and distributors to distribute digital books via a simple catalog format. This format is designed to work interchangeably across multiple desktop and device software programs.

GitBook is providing an OPDS catalog at: `https://api.gitbook.com/opds/catalog.atom`

Create and publish a plugin

A GitBook plugin is a node package published on NPM that follow a defined convention.

Structure

package.json

The `package.json` contains general information about your plugin (name, version, description, ...):

```
{
    "name": "gitbook-plugin-mytest",
    "version": "0.0.1",
    "description": "This is my first GitBook plugin",
    "engines": {
        "gitbook": ">1.x.x"
    }
}
```

You can learn more about `package.json` from the NPM documentation.

The **package name** must begin with `gitbook-plugin-` and the **package engines** should contains `gitbook`.

index.js

The `index.js` is main entry point of your plugin:

```
module.exports = {
    // Map of hooks
    hooks: {},

    // Map of new blocks
    blocks: {},

    // Map of new filters
    filters: {}
};
```

Publish your plugin

GitBook plugins are published and installed from NPM.

To publish a new plugin, you need to create an account on npmjs.com then publish it from the command line:

```
$ npm publish
```

Extend blocks

Extending templating blocks is the best way to provide extra functionalities to authors.

The most common usage is to process the content within some tags at runtime. It's like filters, but on steroids because you aren't confined to a single expression.

Defining a new block

Blocks are defined by the plugin, blocks is a map of name associated with a block descriptor. The block descriptor needs to contain at least a `process` method.

```
module.exports = {
    blocks: {
        tag1: {
            process: function(block) {
                return "Hello "+block.body+", How are you?";
            }
        }
    }
};
```

The `process` should return the html content that will replace the tag. Refer to Context and APIs to learn more about `this` and GitBook API.

Handling block arguments

Arguments can be passed to blocks:

```
{% tag1 "argument 1", "argument 2", name="Test" %}
This is the body of the block.
{% endtag1 %}
```

And arguments are easily accessible in the `process` method:

```
module.exports = {
    blocks: {
        tag1: {
            process: function(block) {
                // block.args equals ["argument 1", "argument 2"]
                // block.kwargs equals { "name": "Test" }
            }
        }
    }
};
```

Handling sub-blocks

A defined block can be parsed into different sub-blocks, for example let's consider the source:

```
{% myTag %}
    Main body
    {% subblock1 %}
    Body of sub-block 1
    {% subblock 2 %}
```

```
      Body of sub-block 1
  {% endmyTag %}
```

Extend Filters

Filters are essentially functions that can be applied to variables. They are called with a pipe operator (`|`) and can take arguments.

```
{{ foo | title }}
{{ foo | join(",") }}
{{ foo | replace("foo", "bar") | capitalize }}
```

Defining a new filter

Plugins can extend filters by defining custom functions in their entry point under the `filters` scope.

A filter function takes as first argument the content to filter, and should return the new content. Refer to Context and APIs to learn more about `this` and GitBook API.

```
module.exports = {
    filters: {
        hello: function(name) {
            return 'Hello '+name;
        }
    }
};
```

The filter `hello` can then be used in the book:

```
{{ "Aaron"|hello }}, how are you?
```

Handling block arguments

Arguments can be passed to filters:

```
Hello {{ "Samy"|fullName("Pesse", man=true}} }}
```

Arguments are passed to the function, named-arguments are passed as a last argument (object).

```
module.exports = {
    filters: {
        fullName: function(firstName, lastName, kwargs) {
            var name = firstName + ' ' + lastName;

            if (kwargs.man) name = "Mr" + name;
            else name = "Mrs" + name;

            return name;
        }
    }
};
```

Extend Assets

Extending assets allow plugins to add css or js to the build website or ebook.

Extending website assets

It is being done by listing the assets in the plugin's entry point. The folder `book.assets` will be copied in the final build. `book.css` and `book'js` will be listed in all HTML pages.

```
module.exports = {
    book: {
        assets: './assets/website',
        css: [
            'mystyle.css'
        ],
        js: [
            'myfile.js
        ]
    }
};
```

Extending ebook assets

It's being done in the same way that for website, using `ebook`. Only CSS files can be listed, JavaScript files are not supported.

```
module.exports = {
    ebook: {
        assets: './assets/ebook',
        css: [
            'mystyle.css'
        ]
    }
};
```

Hooks

Hooks is a method of augmenting or altering the behavior of the process, with custom callbacks.

List of hooks

Relative to the global pipeline

> It is recommended using blocks to extend page parsing.

Name	Description	Arguments
`init`	Called after parsing the book, before generating output and pages.	None
`finish:before`	Called after generating the pages, before copying assets, cover, ...	None
`finish`	Called after everything else.	None

Relative to the page pipeline

Name	Description	Arguments
`page:before`	Called before running the templating engine on the page	Page Object
`page`	Called before outputting and indexing the page.	Page Object

Page Object

```
{
    // Parser named
    "type": "markdown",

    // File Path relative to book root
    "path": "page.md",

    // Absolute file path
    "rawpath": "/usr/...",

    // Progress of the page in summary
    "progress": {
        ...
    },

    // Page Content (available in "page:before")
    "content": "# Hello",

    // Page Content splited in sections (available in "page")
    "sections": [
        {
            "content": "<h1>Hello</h1>",
            "type": "normal"
        }
    ]
}
```

Example to add a title

```
{
    "page:before": function(page) {
        page.content = "# Title\n" +page.content;
        return page;
```

```
    }
}
```

Example to replace some html

```
{
    "page": function(page) {
        page.sections[0].content = page.sections[0].content.replace("<b>", "<strong>")
            .replace("</b>", "</strong>");
        return page;
    }
}
```

Asynchronous Operations

Hooks callbacks can be asynchronous and return a promise.

Example:

```
{
    "init": function() {
        return writeSomeFile()
        .then(function() {
            return writeAnotherFile();
        });
    }
}
```

Context and APIs

GitBooks provides different APIs and contexts to plugins. These APIs can vary according to the GitBook version being used, your plugin should specify the `engines.gitbook` field in `package.json` accordingly.

Context for Blocks and Filters

Blocks and filters have access to the same context, this context is bind to the template engine execution:

```
{
    // Current templating syntax
    "ctx": {
        // For example, after a {% set message = "hello" %}
        "message": "hello"
    },

    // Book instance
    "book" <Book>,

    // Current generator
    "generator": "website"
}
```

For example a filter or block function can access the current book using: `this.book`.

Context for Hooks

Hooks only have access to the `<Book>` instance using `this.book`.

Book instance

The `Book` class is the central point of GitBook, it centralize all access methods. This class is defined in book.js.

book.config.get(key, default)

Access to book's configuration (book.json)

```
// Without default value
var title = book.config.get('title');

// With a default value
var websiteStylesheet = book.config.get('styles.website', 'default_value.css');
```

book.formatString(type, str)

Process a markuped string

```
// Format some markdown string
book.formatString('markdown', 'This is **markdown**')
    .then(function(html) { ... });
```