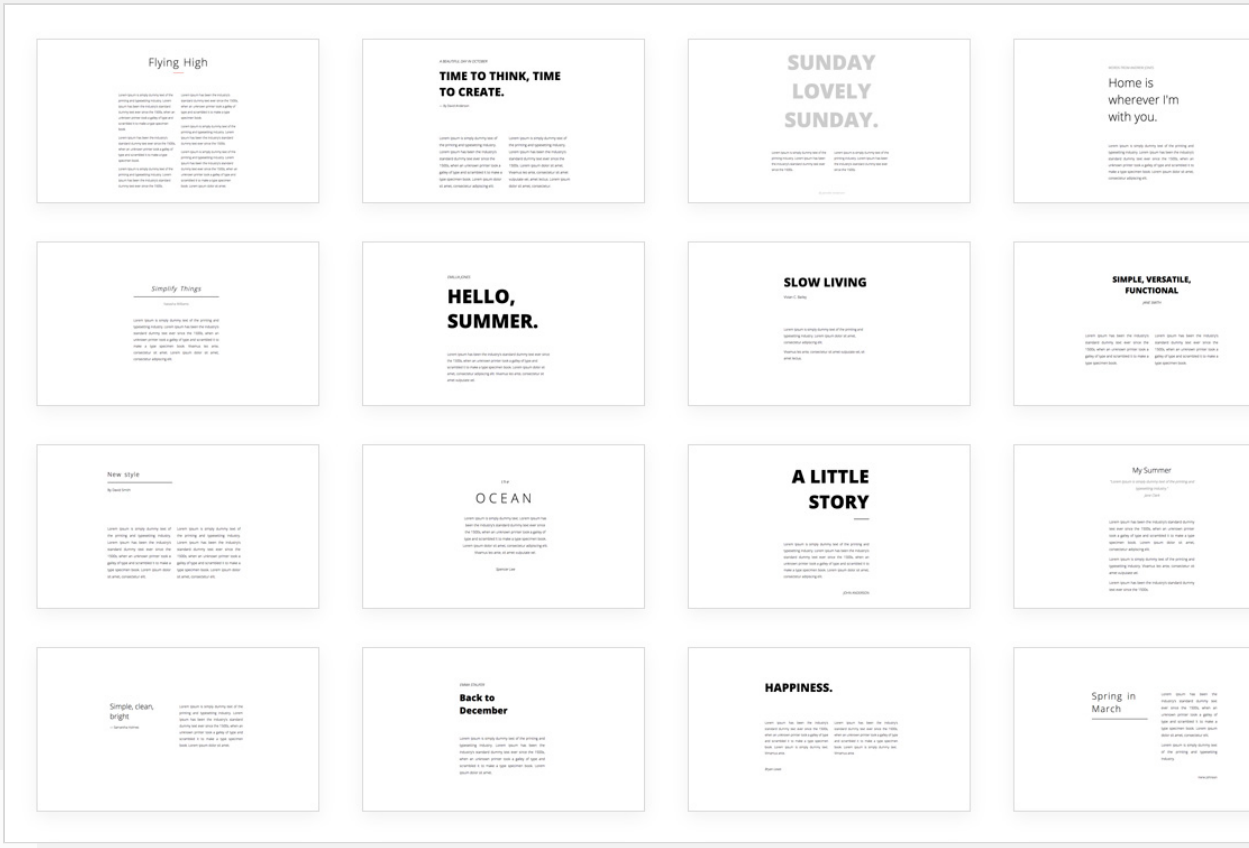
Documentation



InnovaStudio

ContentBuilder.js

ContentBuilder.js is a drag & drop HTML editor javascript library. It is the first editor that allows you to build content layout in almost any css grid framework such as Bootstrap, Tailwind, Foundation and more.

Contents

Getting Started

CSS

Step 1

Include the main css file:

```
<link rel="stylesheet" href="contentbuilder/contentbuilder.css" />
```

and the css file for snippets:

```
<link rel="stylesheet" href="assets/minimalist-blocks/content.css"  />
```

Step 2

JS

Install as Web Library

```
<script src="contentbuilder/contentbuilder.min.js" type="text/javascript">
```

Or Install with NPM

```
npm install @innovastudio/contentbuilder
```

Then import into your project:

```
import ContentBuilder from '@innovastudio/contentbuilder';
```

Usage

```
<div class="container">
</div>
```

```
const builder = new ContentBuilder({
    container: '.container'
});
```

To get the HTML:

```
let html = builder.html();
```

Then you can do anything with the html, for example, posting it to the server for saving, etc.

Examples (HTML, PHP, React, Vue)

ContentBuilder.js is written in pure Javascript (ES6) so you can use it in most situations. Sample use in simple HTML, PHP, React and Vue projects are included.

React and Vue project examples are provided in separate downloads in the user area.

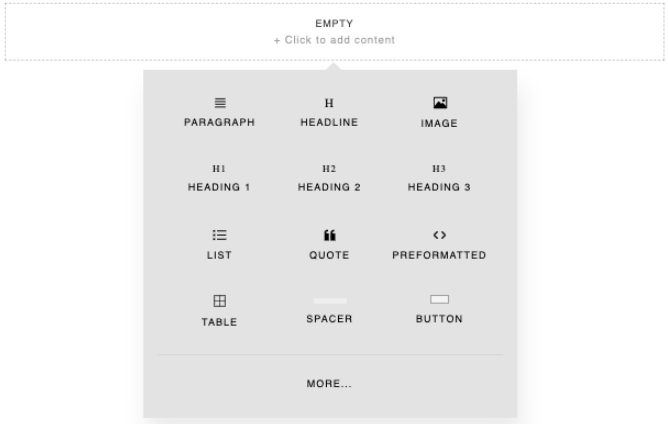
Folder Structure

- public/
  - assets/
  - contentbuilder/
  - uploads/
  - example1.html (HTML example)
  - example2.php (PHP example)
  - example3.html (Multiple editable area example)
  - ...Other examples
- src/
  - contentbuilder/ (Only provided in Source Code package)
  - scss/ (Only provided in Source Code package)
- docs/
- README.md (Documentation)
- readme.txt (Readme file)
- readme-sourcecode.txt (Readme file for Source Code package)
- ...

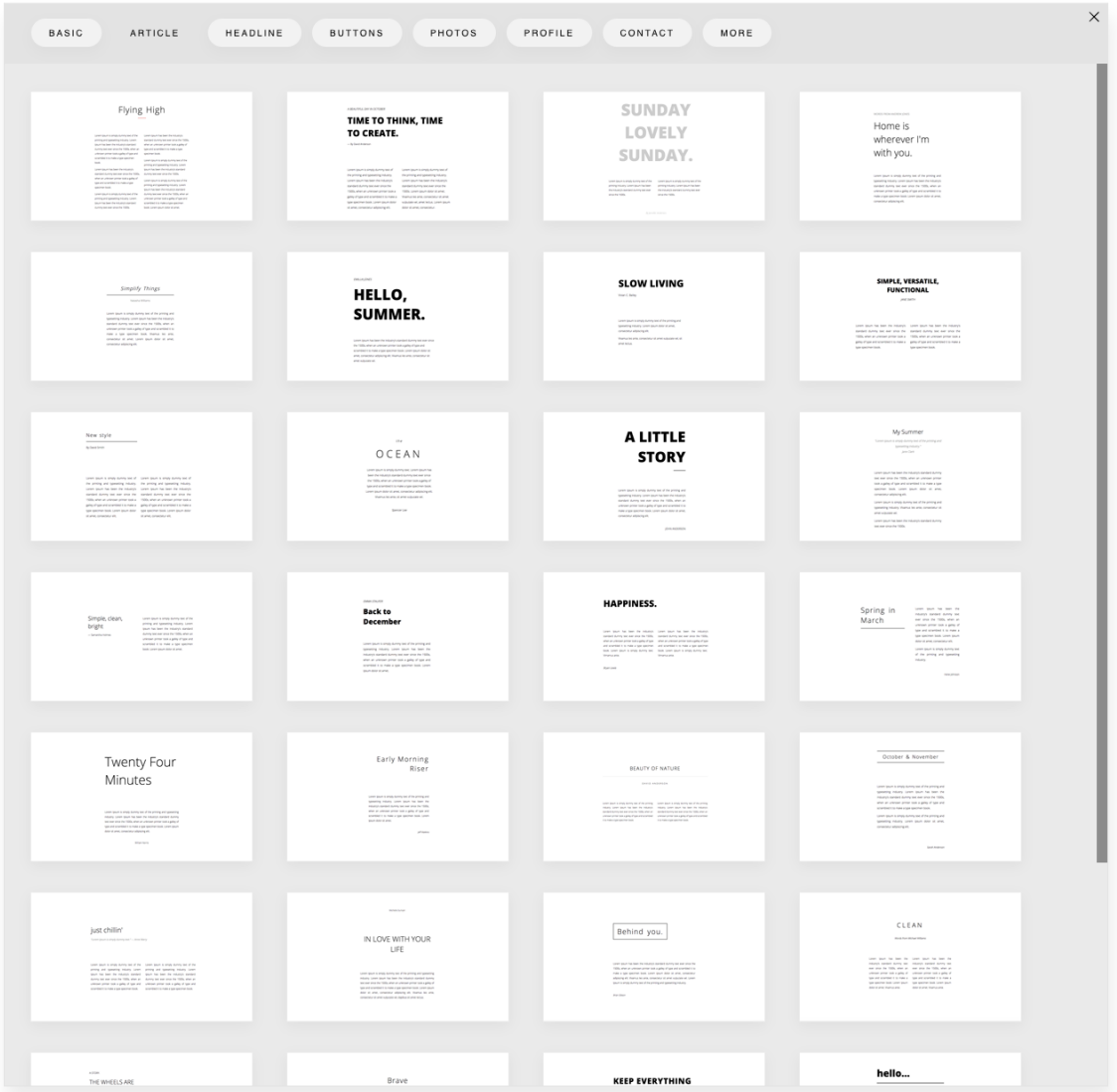
Snippets

Snippets are predesigned blocks that you can add or drag & drop into your content.

By default, snippets are enabled. When you add a content, please click the MORE button to open the snippets dialog.



Snippet dialog opened.



Snippet files are located in the folder:

assets/minimalist-blocks/

It contains:

- content.js (snippets file)
- content.css (snippet style)
- images

You can configure the snippets location by setting the **snippetUrl** and **snippetPath** parameters:

```
const builder = new ContentBuilder({
    container: '.container',
    snippetUrl: 'assets/minimalist-blocks/content.js', // Snippet file
    snippetPath: 'assets/minimalist-blocks/',  // Location of snippets' assets
});
```

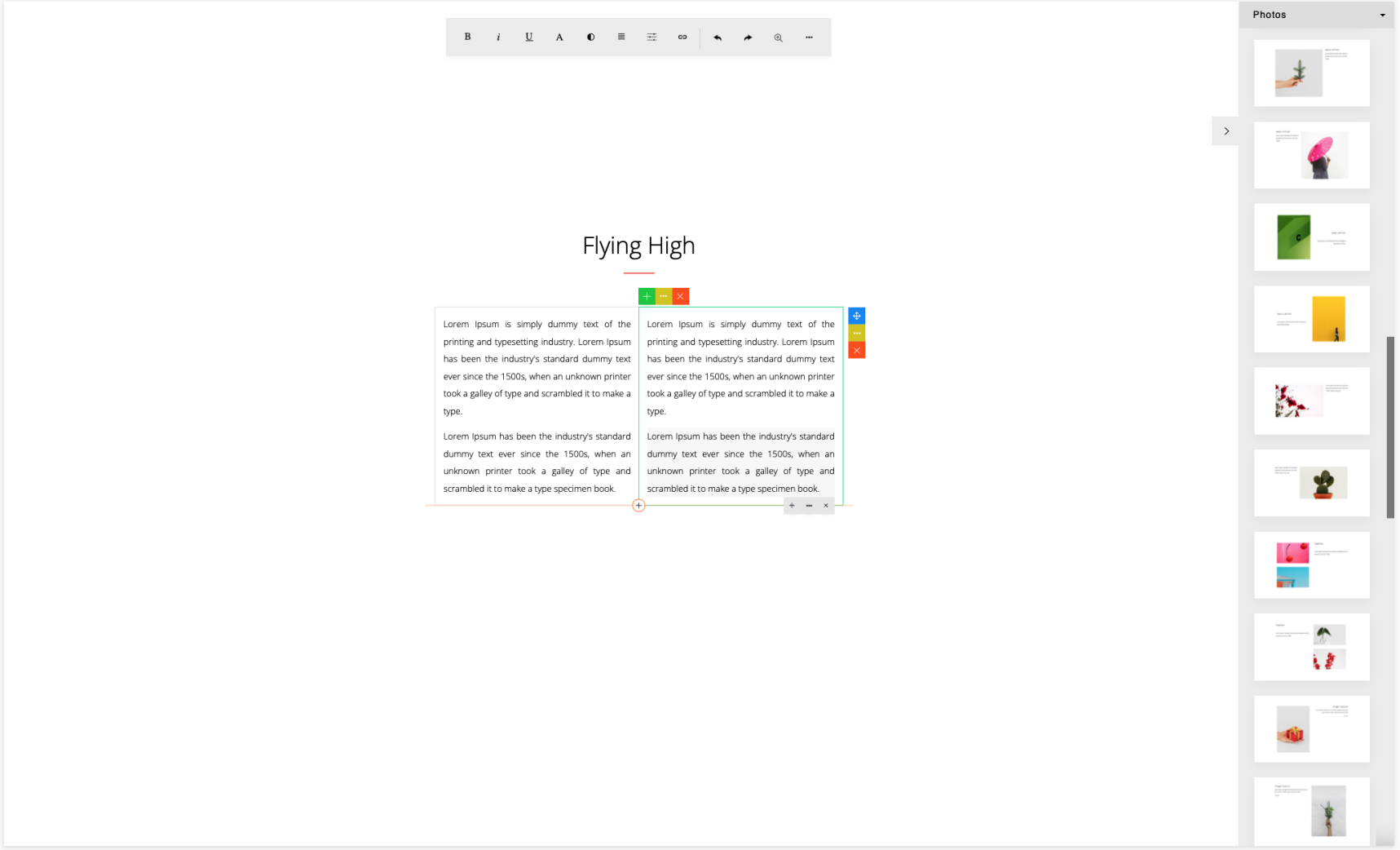
In case of a different location, path adjustment may be needed. Here you can use the **snippetPathReplace** parameter.

```
const builder = new ContentBuilder({
    ...
    snippetPathReplace: ['assets/minimalist-blocks/', 'mycustomfolder/assets/minimalist-blocks/'],
    ...
});
```

In this example, the default location is changed to **mycustomfolder/assets/minimalist-blocks/**

Snippet Side Panel

Snippets can be displayed on a side panel. This allows you to drag & drop snippets into your content.



By default, this feature is not enabled.

To enable this feature, include the snippet file on the page:

```
<script src="assets/minimalist-blocks/content.js" type="text/javascript"></script>
```

Or load it programmatically:

```
builder.loadSnippets('assets/minimalist-blocks/content.js');
```

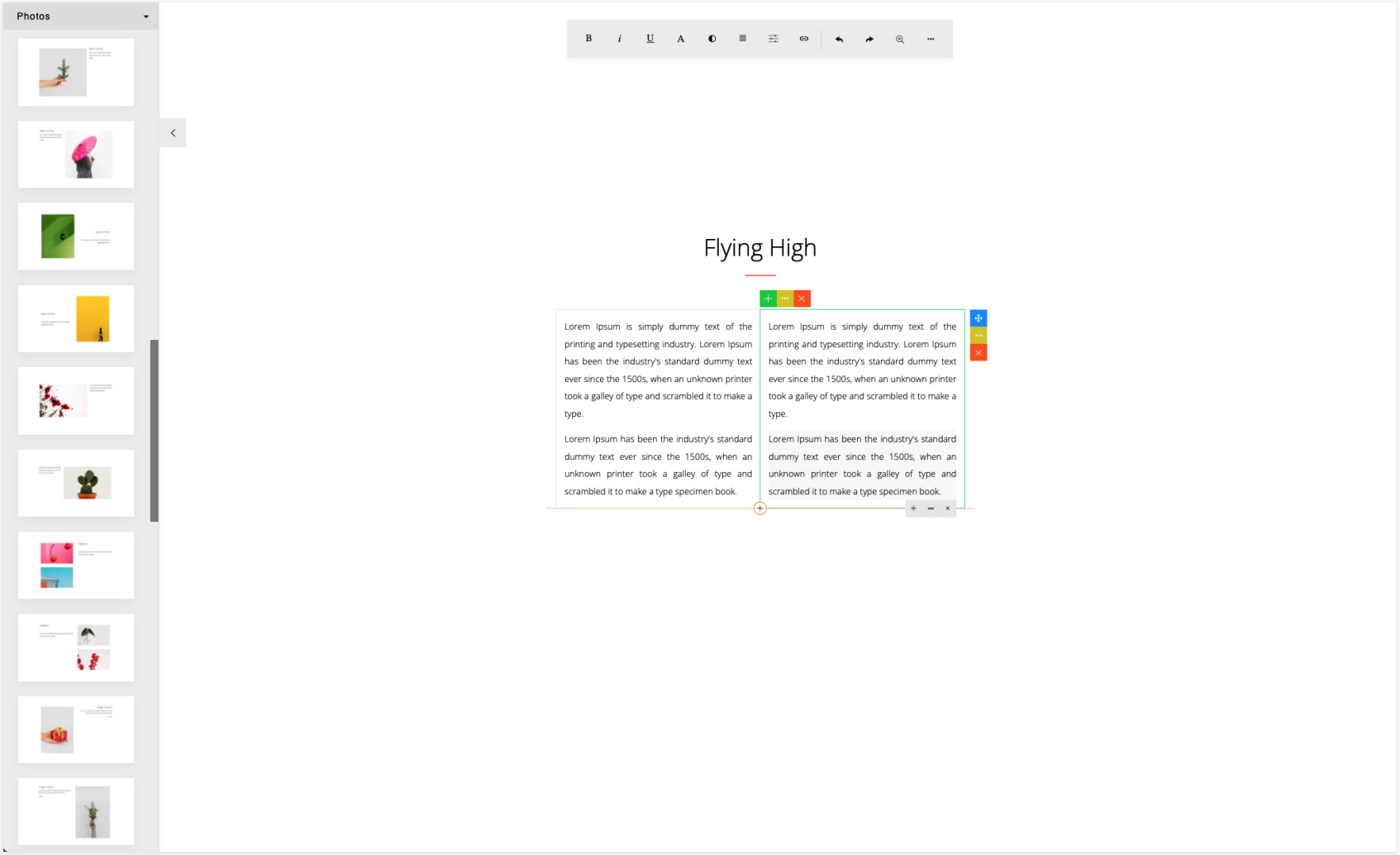
Configuration

Here are some options to configure the snippet side panel:

Placement

```
const builder = new ContentBuilder({
    container: '.container',
    sidePanel: 'left'
});
```

The default placement is ‘right’. If set to ‘left’, the placement will be on the left as seen below.



Open on the first load

By default, the snippet side panel is closed. To open it on the first load:

```
const builder = new ContentBuilder({
    container: '.container',
    snippetOpen: true
});
```

Display mode

If the snippet side panel is not used (during editing), it will be automatically closed. To make it always visible:

```
const builder = new ContentBuilder({
    container: '.container',
    snippetDisplay: 'visible' // default: auto
});
```

Snippet Categories

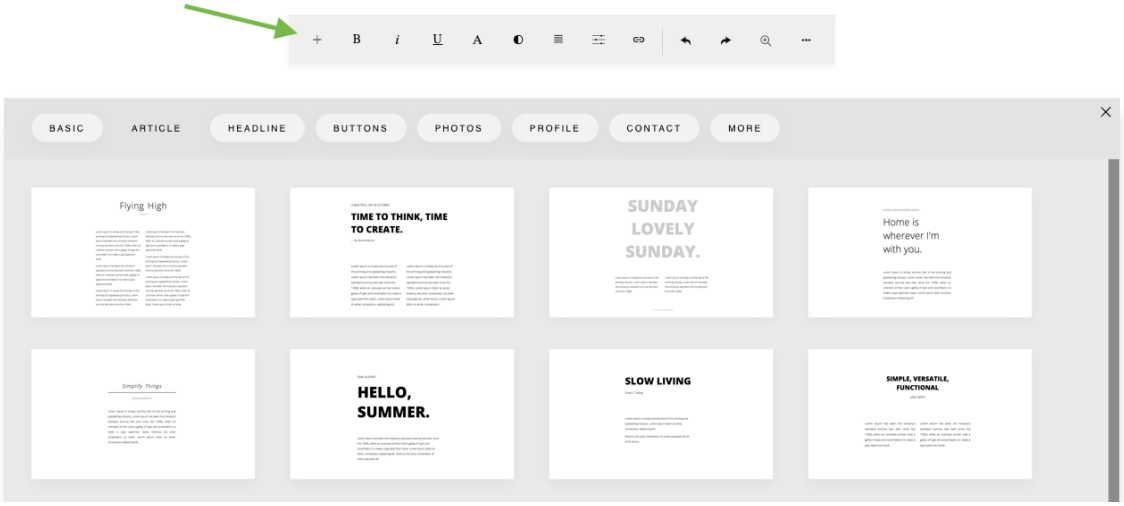
Snippets have multiple categories for different purposes. The built-in categories are defined in the **snippetCategories** parameter:

```
const builder = new ContentBuilder({
    container: '.container',
    snippetCategories: [
        [120,'Basic'],
        [118,'Article'],
        [101,'Headline'],
        [119,'Buttons'],
        [102,'Photos'],
        [103,'Profile'],
        [116,'Contact'],
        [104,'Products'],
        [105,'Features'],
        [106,'Process'],
        [107,'Pricing'],
        [108,'Skills'],
        [109,'Achievements'],
        [110,'Quotes'],
        [111,'Partners'],
        [112,'As Featured On'],
        [113,'Page Not Found'],
        [114,'Coming Soon'],
        [115,'Help, FAQ']
    ],
    defaultSnippetCategory: 120, // the default category is 'Basic'
});
```

The **defaultSnippetCategory** specifies the category to open on the first load.

Snippet Button

You can enable the Snippet button on the toolbar for quick access to the snippets.



Here you can open the Snippet dialog from the toolbar by clicking the (+) button.

To enable the button:

```
const builder = new ContentBuilder({
    container: '.container',
    toolbarAddSnippetButton: true
});
```

Please see the Toolbar section for more customization on the toolbar.

Programmatically Open Snippet Dialog

If needed, you can open the Snippet dialog programmatically by calling the **viewSnippets()** method:

```
builder.viewSnippets();
```

Language File

With the Language file, you can translate the ContentBuilder.js interface into another language.

The provided language files:

- contentbuilder/lang/en.js
- contentbuilder/lang/fr.js

To enable the language file, you need to add the file before including ContentBuilder.js:

```
<script src="contentbuilder/lang/en.js" type="text/javascript"></script>
```

or

```
<script src="contentbuilder/lang/fr.js" type="text/javascript"></script>
```

Here is the language file content as seen on **en.js**:

```
var _txt = new Array();
_txt['Bold'] = 'Bold';
_txt['Italic'] = 'Italic';
```

You can create your own language file by copying/modifying this file.

Icon Support

ContentBuilder.js allows you to insert icons. Some snippets are designed using icons as well. Icons are located in the assets folder:

assets/ionicons/

You can see the icons that are imported in the snippet css file (assets/minimalist-blocks/content.css):

```
@import url("../ionicons/css/ionicons.min.css");
```

You can modify this if you want to change the location of the icons.

Note that icons need to be located inside the assets folder. You can change the assets folder location by setting:

```
const builder = new ContentBuilder({
    container: '.container',
    assetPath: 'assets/',
});
```

Custom File/Media Browser or Asset Manager Support

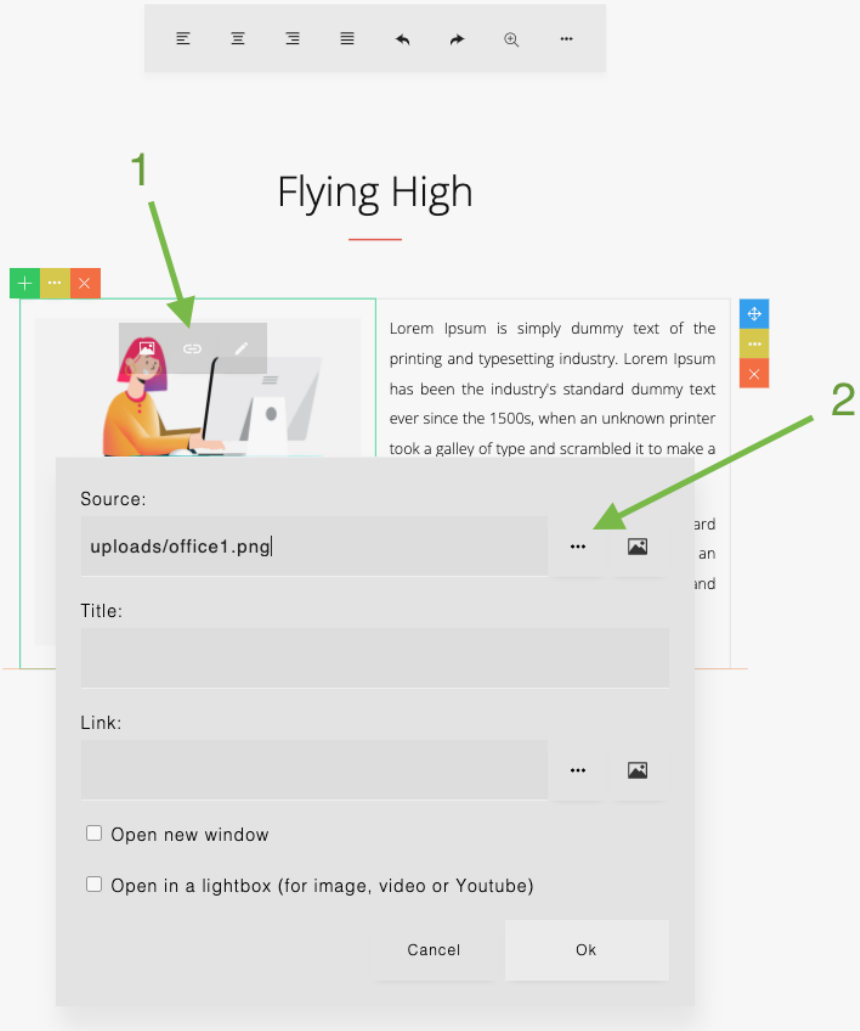
You can specify your own custom file/media browser (or Asset Manager) to work with ContentBuilder.js.

As an example, a simple file/media browser (**assets.html**) is provided with some sample images or videos that you can select and insert into your content.

To configure:

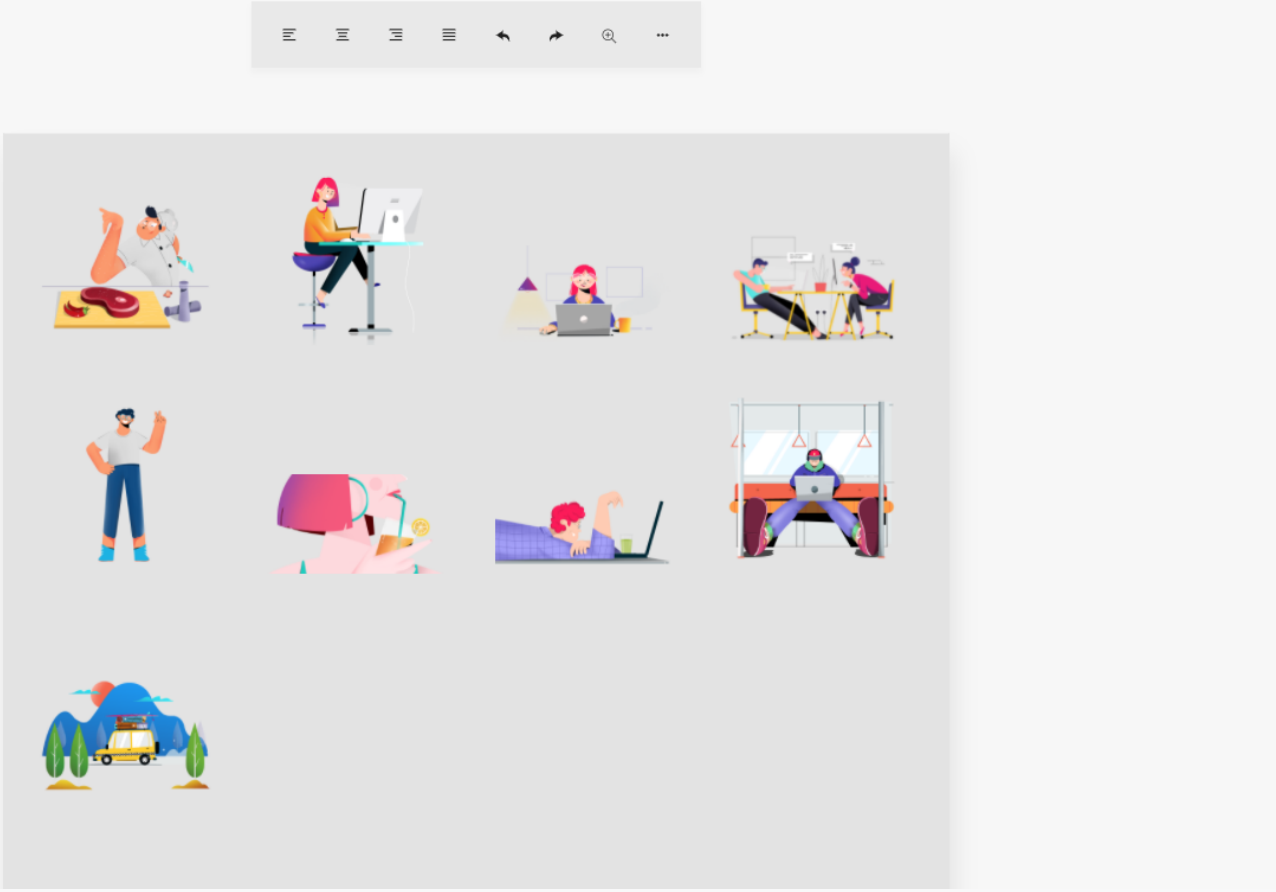
```
const builder = new ContentBuilder({
    container: '.container',
    imageSelect: 'assets.html',
    fileSelect: 'assets.html',
    videoSelect: 'assets.html'
});
```

To try, click the link icon (1) on a selected image to open the image dialog. On the image dialog, you see a browse button displayed (2).



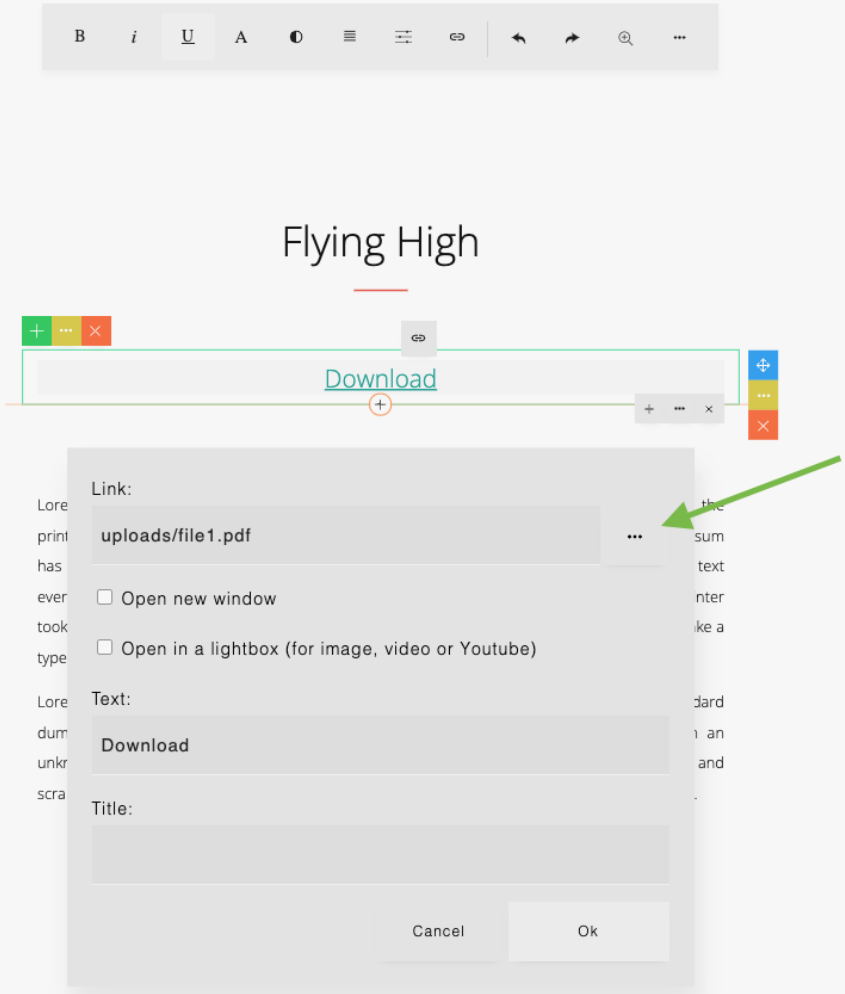
Click the browse button to open the file/media browser as specified (the assets.html).

File/media browser opened:



Then you can select an image.

The file/media browser can also be opened from the link dialog and video settings dialog.



Click the browse button to open the file/media browser as specified (the assets.html).

Please see assets.html if you want to create your own file/image browser. Use the **selectAsset()** function as shown in assets.html to work with ContentBuilder.js. So when you select a file or image, the url will be returned to ContentBuilder.js.

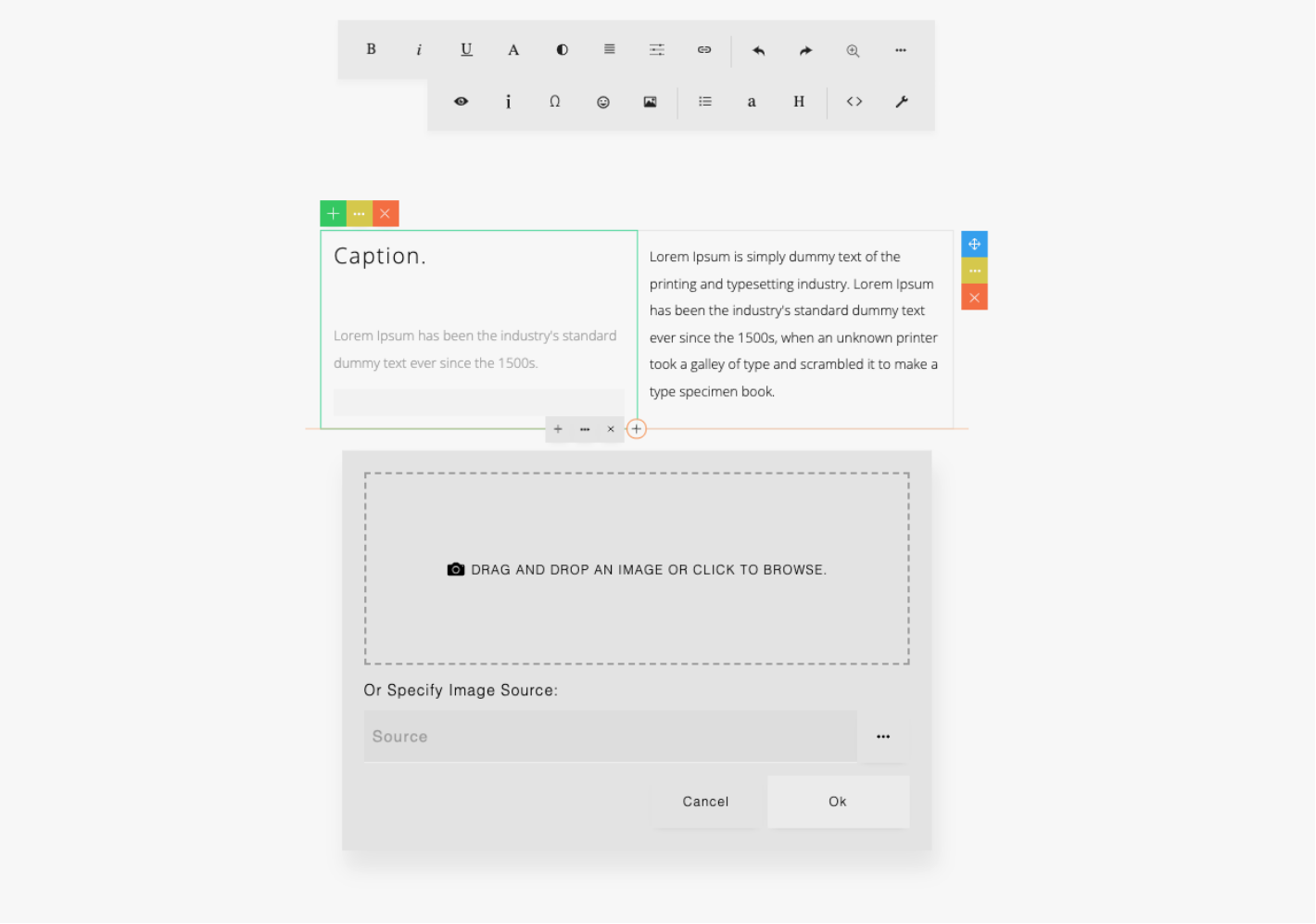
Using Custom Function

You can replace the **imageSelect**, **fileSelect** and **videoSelect** parameters with the **onImageSelectClick**, **onFileSelectClick** and **onVideoSelectClick** parameters that will run your own custom function.

```
const builder = new ContentBuilder({
    container: '.container',
    onImageSelectClick: function () { ... },
    onFileSelectClick: function () { ... },
    onVideoSelectClick: function () { ... },
});
```

Saving Base64 Images

When you insert an image or change an embedded image, you will be able to browse the local image file. The selected image will be embedded as a base64 format in your content, so you will see the result immediately without having to wait for the image upload process.



The embedded image will be:

```
<img src="....">
```

You can save all embedded base64 images to the server by calling the **saveImages()** method. Normally, you'd call the **saveImages()** method on your saving implementation. So before saving the HTML content, all embedded images will be saved first on the server (ex. as jpg, png) and all base64 image sources will be replaced (automatically) with the physical image url. Then the HTML content can be saved. This process is made simple by using the **saveImages()** method.

There are 2 ways to use the **saveImages()** method:

1. Using Form Method

Example (if you're using PHP):

```
builder.saveImages('saveimage.php', function(){

    // Image saving done. Then you can save the content
    let html = builder.html();

    // ...

});
```

Here you specify a server side handler (**saveimage.php**) that performs the file saving on the server. File saving on the server is not part of ContentBuilder.js and depends on your server environment. As examples, you can use the provided handler:

- saveimage.php (for PHP)
- saveimage.ashx (for ASP.NET)

Please add your own security handling on these files.

To try a working example, please open **example2.php**.

2. Using AJAX Post Method

```
builder.saveImages('', ()=>{

    // Image saving done. Then you can save the content
    let html = builder.html();

    // ...

}, (img, base64, filename)=>{

    // Upload base64 images
    const reqBody = { image: base64, filename: filename };
    fetch('/uploadbase64', {
        method:'POST',
        body: JSON.stringify(reqBody),
        headers: {
            'Content-Type': 'application/json',
        }
    })
    .then(response=>response.json())
    .then(response=>{
        if(!response.error) {
            const uploadedImageUrl = response.url;
            img.setAttribute('src', uploadedImageUrl); // Update image src
        }
    });

});
```

In this example, each image will be sent to an endpoint: **/uploadbase64**

If you're using Node.js, you can implement the endpoint to save the images using:

```
const express = require('express');
const fs = require('fs');
const app = express();
const path = require('path');
const cors = require('cors');
const serveStatic = require('serve-static');
const formidable = require('formidable-serverless');
const sharp = require('sharp');

var $path =  __dirname + '/public/uploads/'; // Physical path
var $urlpath = 'uploads/'; // URL path

app.use(cors());
app.use(express.urlencoded({
    extended: true
}));
app.use(express.json({ limit: '50mb' }));
app.use(serveStatic(path.join(__dirname, '/public')));

app.post('/uploadbase64', async (req, res) => {
    const base64Data = req.body.image;
    const filename = req.body.filename;

    let extension = path.extname(filename).toLowerCase();

    let imageFile=base64Data;
    if(extension  === '.jpeg' || extension === '.jpg') {
        let img = new Buffer.from(base64Data, 'base64');
```

```
    fs.writeFile($path + filename, imageFile, 'base64', async (err)=>{
        if (err) {
            res.status(500).json({
                ok:true, status: 500,
                error: 'Something went wrong.'
            });
            return
        }
        res.status(200).json({
            ok:true, status: 200,
            url:  $urlpath + filename
        });
    });
});

app.listen(8081, function() {
    console.log('App running on port 8081');
});
```

Example use of the AJAX post method can be seen on:

- React Example (react-contentbuilder folder, please see src/components/contentbuilder/buildercontrol.jsx)
- Vue Example (vue-contentbuilder folder, please see src/components/Editable.vue)
- src/index.js (in Source Code package)

Configuration

Here are some options you can set to configure the image embed functionality:

Image Quality

```
const builder = new ContentBuilder({
    container: '.container',
    imageQuality: 0.92,
});
```

Maximum Width

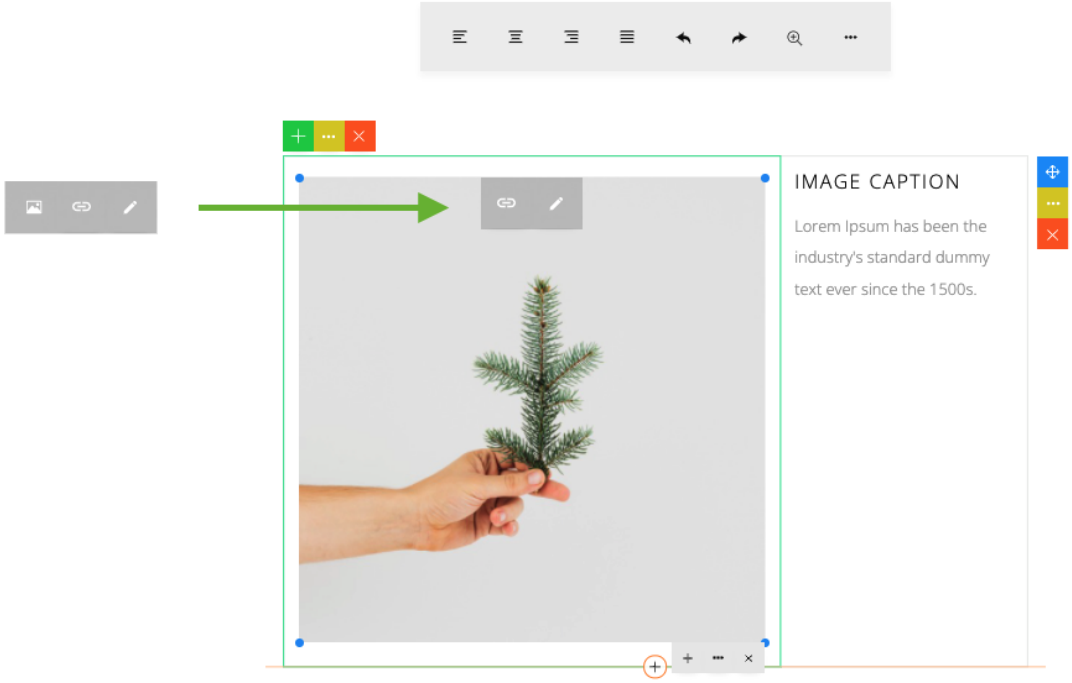
```
const builder = new ContentBuilder({
    container: '.container',
    maxEmbedImageWidth: 1600,
});
```

If maxEmbedImageWidth is set to -1, then no maximum width will be applied (will use original image width).

Enable/Disable Embed Image Feature.

```
const builder = new ContentBuilder({
    container: '.container',
    imageEmbed: false,
});
```

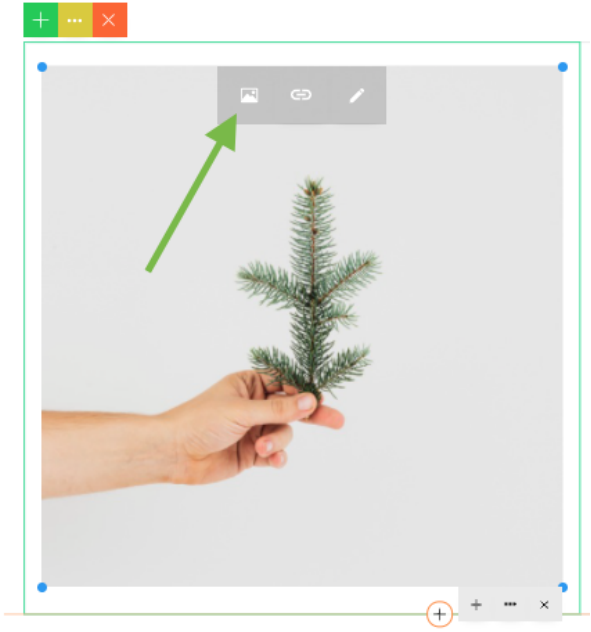
If imageEmbed is set to false, the browse image button will not be displayed as seen below.



Using Custom Function on Browse Image Click

```
const builder = new ContentBuilder({
    container: '.container',
    onImageBrowseClick: function() { ... },
});
```

The function will be called when the browse image button is clicked.



Using Custom Function on Image Link/Settings Click

```
const builder = new ContentBuilder({
    container: '.container',
    onImageSettingClick: function() { ... },
});
```

The function will be called when the image settings button is clicked.

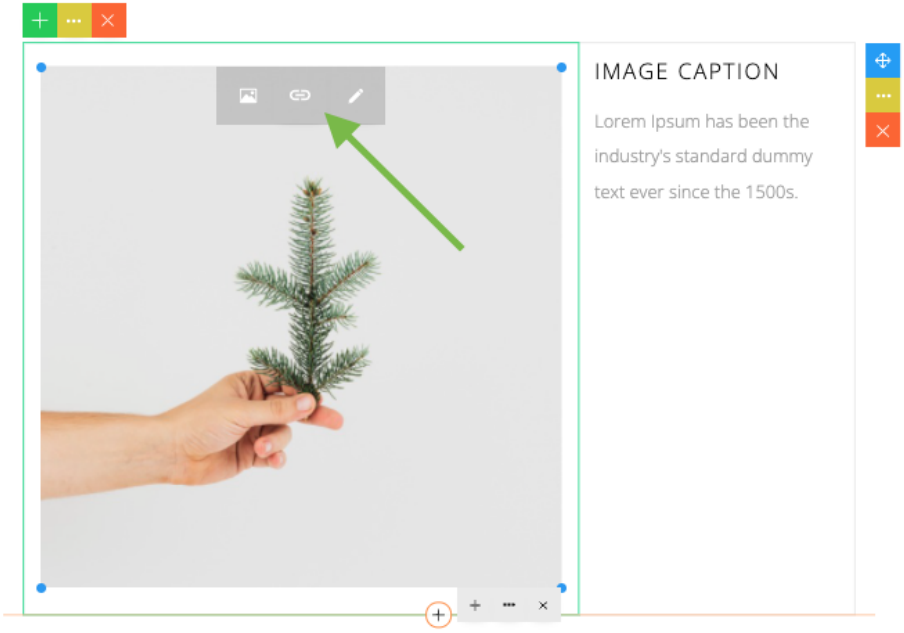
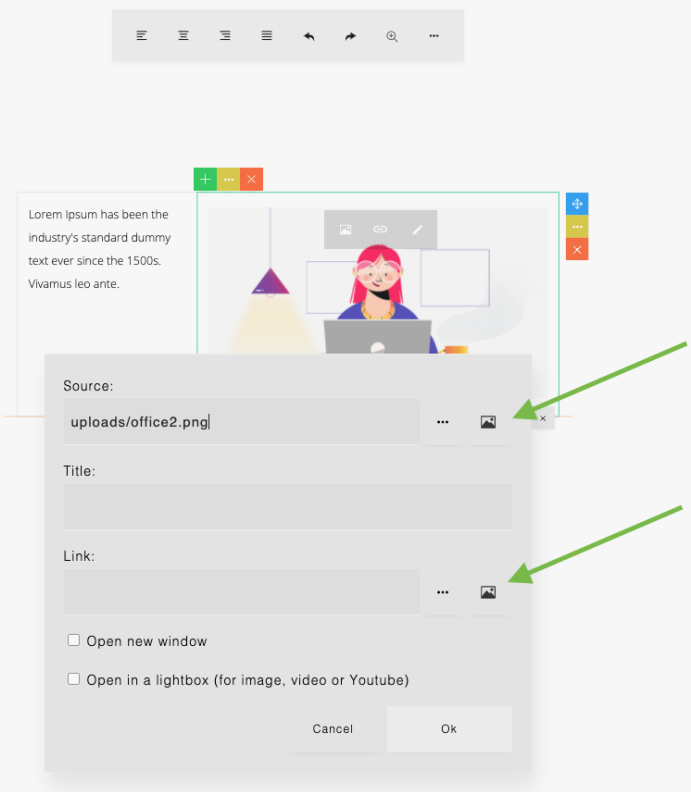
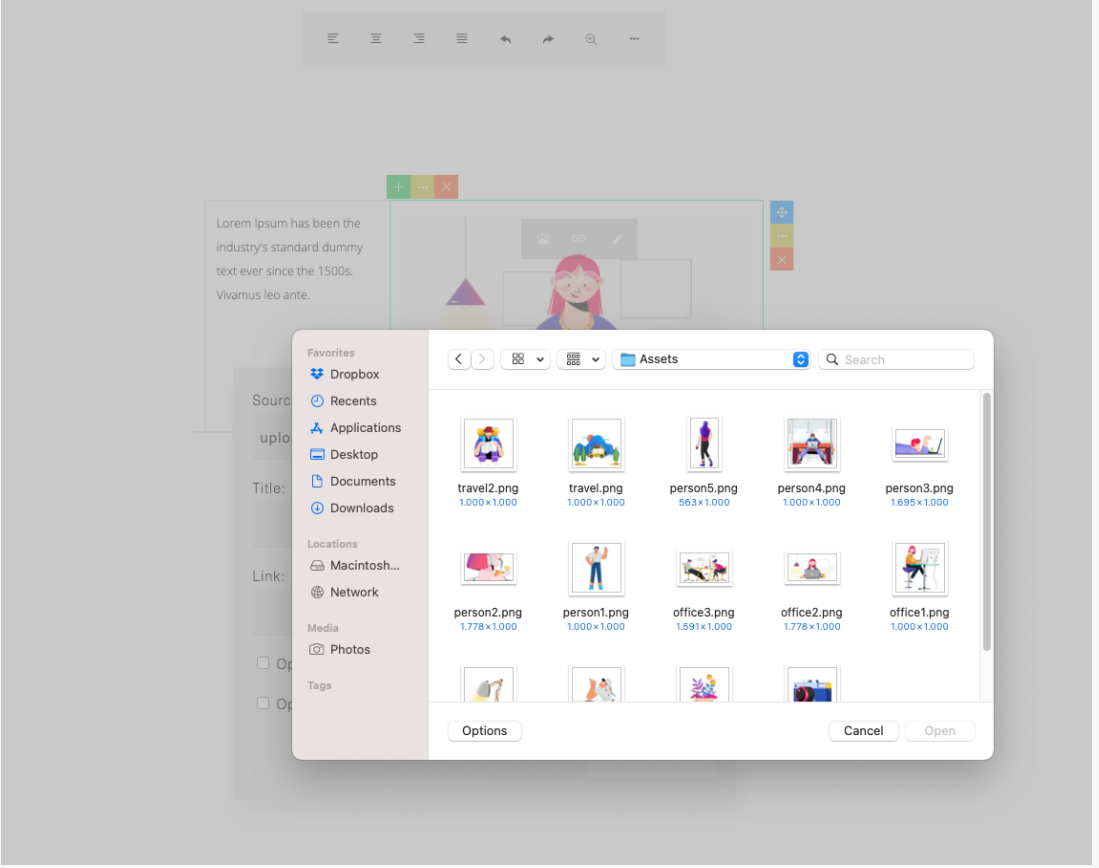


Image Upload from the Image Dialog

A browse local image button can be enabled on the Image dialog. This can be used to select and upload image to the server.





There are 2 ways to enable the button:

1. Using Form Method

Use the **mediaHandler** parameter. You can also use the **largerImageHandler** parameter (previous version) which is the same and still works.

```
const builder = new ContentBuilder({
    container: '.container',
    mediaHandler: 'savemedia.php' // or savemedia.ashx if you're using ASP.NET
});
```

Here you specify a server side handler for uploading files.

As examples, you can use the provided handler:

- savemedia.php (for PHP)
- savemedia.ashx (for ASP.NET)

By default, images are saved in the **upload/** folder. You can change the upload folder by editing **savemedia.php** or **savemedia.aspx**.

2. Using AJAX Post Method

Use the **onMediaUpload** parameter. You can also use the **onLargerImageUpload** parameter (previous version) which is the same and still works.

```
const builder = new ContentBuilder({
    container: '.container',
    onMediaUpload: (e)=>{
        uploadFile(e, (response)=>{
        if(!response.error) {
            const uploadedImageUrl = response.url;
            if(uploadedImageUrl) obj.returnUrl(uploadedImageUrl);
        }
        });
    },
});

function uploadFile(e, callback) {

    const selectedFile = e.target.files[0];

    const formData = new FormData();
    formData.append('file', selectedFile);
    fetch('/upload', {
        method: 'POST',
        body: formData,
    })
    .then(response=>response.json())
```

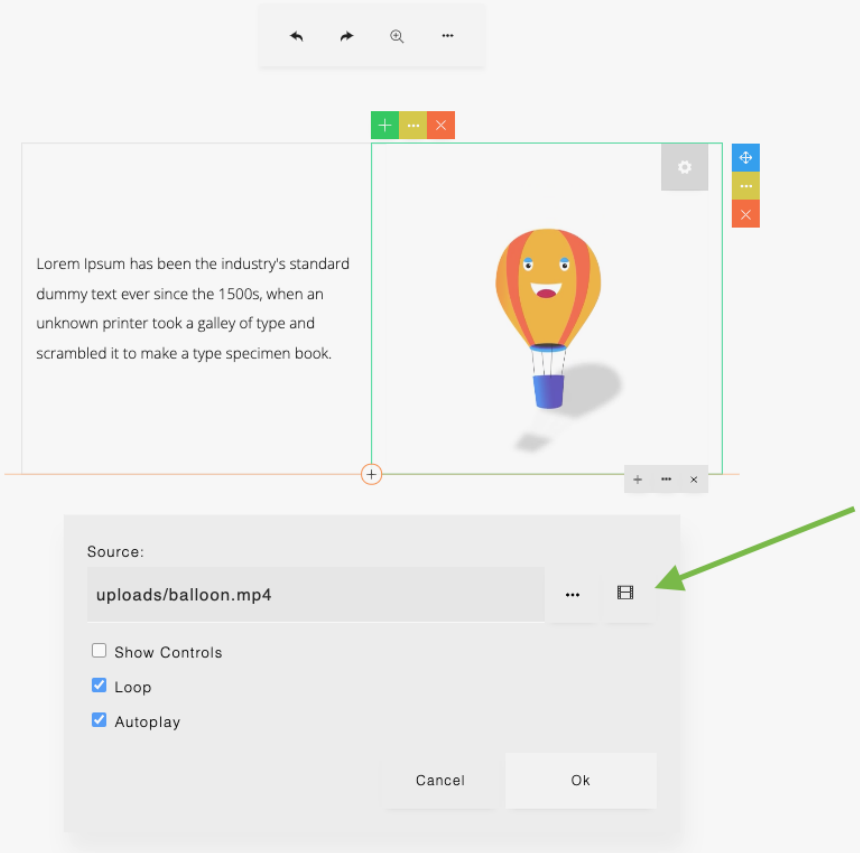
The example above uses the **returnUrl(url)** method to apply the uploaded image url back to the image dialog. You can also use the **applyLargerImage** parameter (previous version) which is the same and still works.
In the example, file will be sent to an endpoint: **/upload**

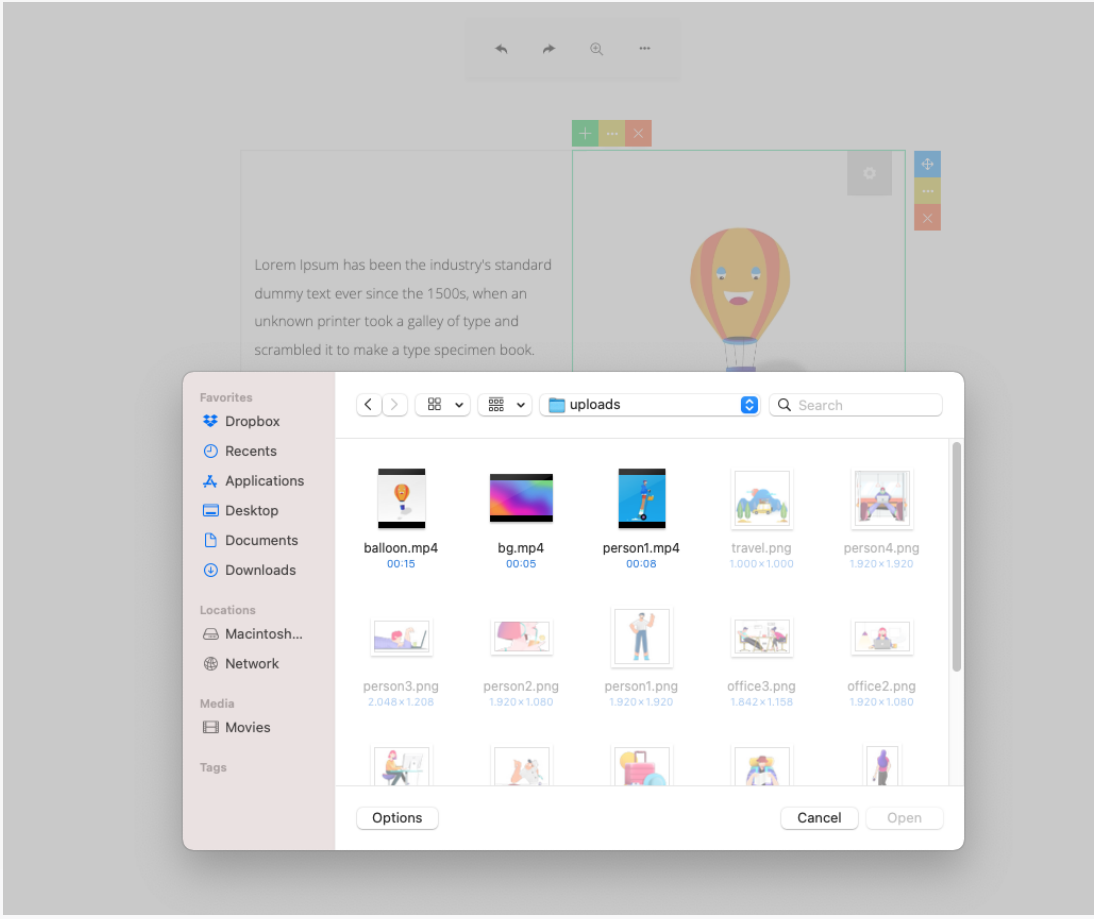
Working examples can be seen on the React & Vue examples:

- React Example (react-contentbuilder folder, please see src/components/contentbuilder/buildercontrol.jsx)
- Vue Example (vue-contentbuilder folder, please see src/components/Editable.vue)
- src/index.js (in Source Code package)

Video Upload from the Video Dialog

A browse local video button can be enabled on the video dialog. This can be used to select and upload video (mp4 file) to the server.





There are 2 ways to enable the button:

1. Using Form Method

Use the **videoHandler** parameter.

```
const builder = new ContentBuilder({
    container: '.container',
    videoHandler: 'savemedia.php' // or savemedia.ashx if you're using ASP.NET
});
```

Here you specify a server side handler for uploading files.

As examples, you can use the provided handler:

- savemedia.php (for PHP)
- savemedia.ashx (for ASP.NET)

By default, videos are saved in the **upload/** folder. You can change the upload folder by editing savemedia.php or savemedia.aspx.

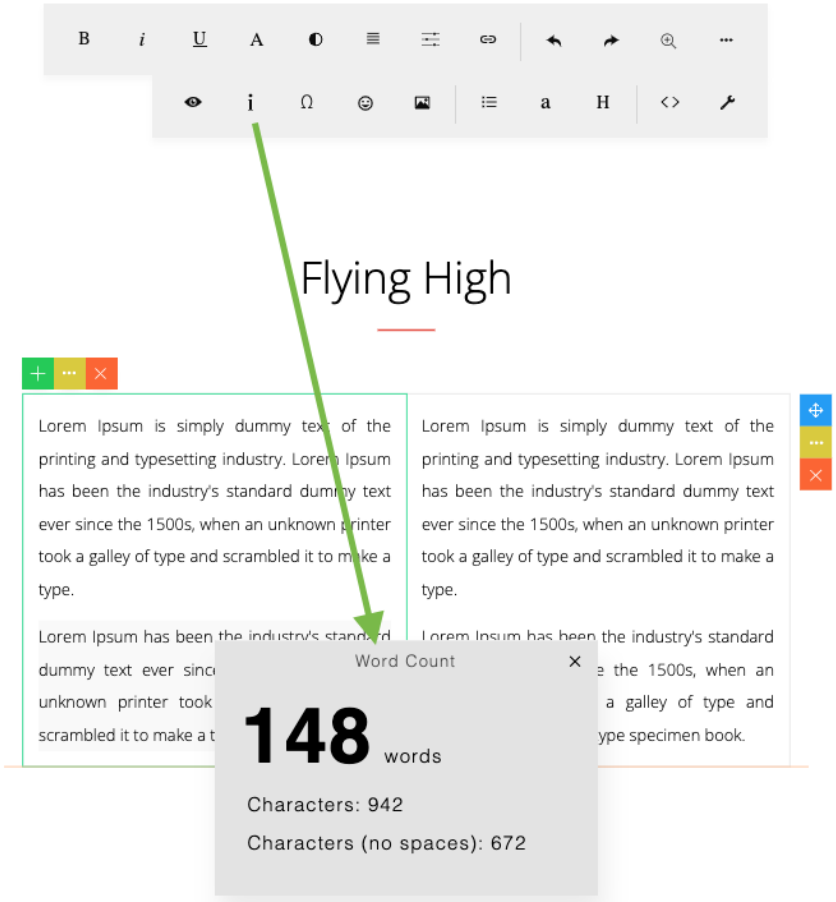
2. Using AJAX Post Method

Use the **onVideoUpload** parameter.

```
const builder = new ContentBuilder({
    container: '.container',
    ...
    onVideoUpload: (e)=>{
        uploadFile(e, (response)=>{
        if(!response.error) {
            const uploadedFileUrl = response.url;
            if(uploadedFileUrl) obj.returnUrl(uploadedFileUrl);
        }
        });
    },
});

function uploadFile(e, callback) {

    const selectedFile = e.target.files[0];

    const formData = new FormData();
    formData.append('file', selectedFile);
    fetch('/upload', {
        method: 'POST',
        body: formData,
    })
    .then(response=>response.json())
```

Plugins

With plugins, you can extend the ContentBuilder.js features. A plugin can add an additional button into the ContentBuilder.js toolbar. For example, a 'wordcount' plugin adds a button in the toolbar and when clicked, a dialog showing the word count info will be diplayed.



Plugin scripts are located in the **plugins/** folder inside **contentbuilder/**.

You will see the folder containing some prebuilt plugins:

- contentbuilder/plugins/helloworld
- contentbuilder/plugins/preview
- contentbuilder/plugins/searchreplace
- contentbuilder/plugins/showgrid
- contentbuilder/plugins/symbols
- contentbuilder/plugins/wordcount

To load the plugins, you can configure:

```
const builder = new ContentBuilder({
    container: '.is-container',
    plugins: [
        { name: 'preview', showInMainToolbar: true, showInElementToolbar: true },
        { name: 'wordcount', showInMainToolbar: true, showInElementToolbar: true },
        { name: 'symbols', showInMainToolbar: true, showInElementToolbar: false },
    ],
    pluginPath: 'contentbuilder/',
});
```

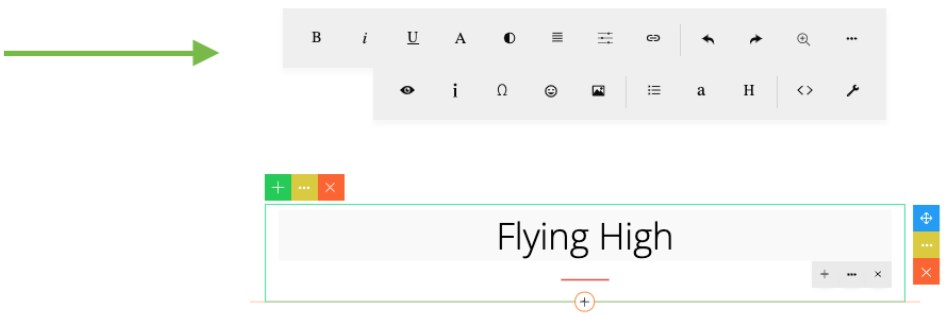
Here we specify the **plugins** and **pluginPath** parameters.

The plugins parameter accepts list of plugins with configurations:

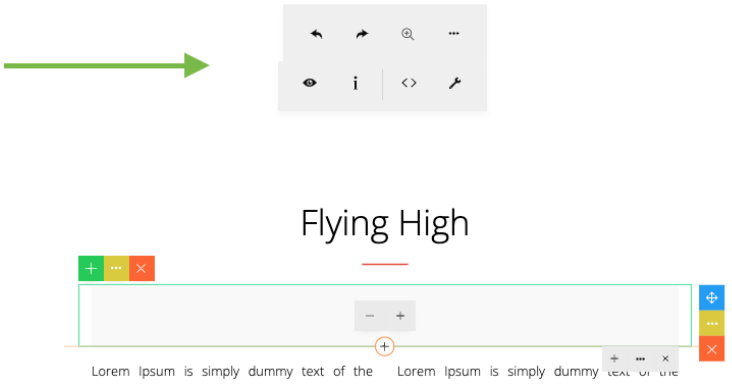
- name: name of the plugin (same as the folder name)
- showInMainToolbar: the plugin will add a button on the main editing toolbar (when text element is selected).
- showInElementToolbar: the plugin will add a button on the element editing toolbar (when non-text element is selected).

The difference between the main editing toolbar and the element editing toolbar is that the main editing toolbar shows when a text element is selected (cursor is placed on text), whereas the element editing toolbar shows when a non-text element is clicked/selected (such as when selecting an image, spacer, etc).

Main editing toolbar (text-based):

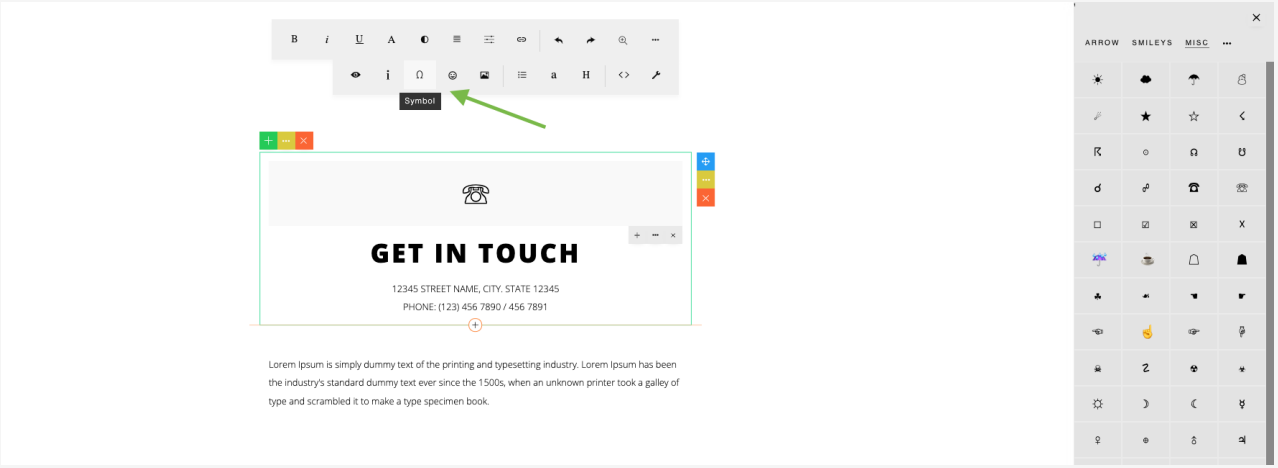


Element editing toolbar (non text):

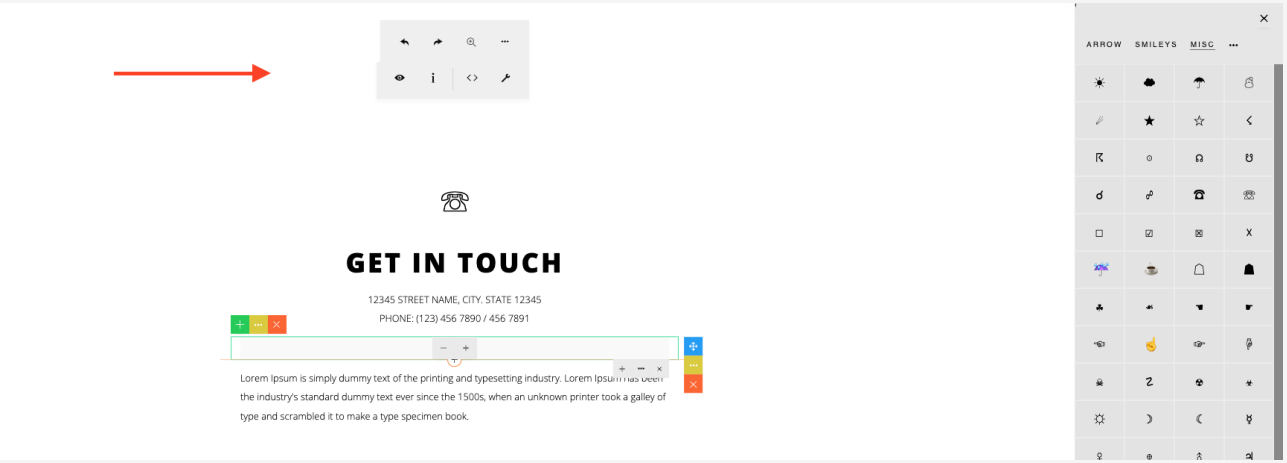


As seen in the example configuration above:

The ‘symbols’ plugin should display its button only if a text element is selected/clicked. So the **showInMainToolbar** (main editing toolbar) is set to true and **showInElementToolbar** (element editing toolbar) is set to false. Here the Symbol button is displayed on the main editing toolbar:



Here the Symbol button is not displayed on the element editing toolbar:



In the previous version, there is a ‘buttoneditor’ plugin which shows an edit (pencil) icon when a link button is clicked. This plugin is now integrated with the builder, so no configuration is needed.

An alternative way to load the plugins is by using a configuration file:

contentbuilder/config.js

If the plugins parameter is not set during initialization (as in the above example),
ContentBuilder.js will automatically load this configuration file. Sample configuration in **config.js**:

```
_cb.settings.plugins = [
    { name: 'preview', showInMainToolbar: true, showInElementToolbar: true },
    { name: 'wordcount', showInMainToolbar: true, showInElementToolbar: true },
    { name: 'symbols', showInMainToolbar: true, showInElementToolbar: false },
];
```

To disable loading this file, you can set **disableConfig** to false:

```
const builder = new ContentBuilder({
    container: '.is-container',
    disableConfig: false
});
```

Note that using an external configuration file is an approach that comes from the previous version of ContentBuilder.js.
In the previous version, the plugins parameter is configured like this:

```
['preview','wordcount']
```

This still works, however we recommend using the new approach (explained above) for future flexibility.

Plugin Development

If you want to create your own plugin, please see this guide:

https://innovastudio.com/contentbuilder-plugins

Modules

With modules, you can extend snippets.

Normally, snippets are defined as static HTML. With a module snippet (snippet as a module), you can a create dynamic (non-static) snippets for ContentBuilder.js. You can add custom scripts to control the snippet content, create interactive elements and make it configurable via a custom dialog.

To create your own module, please see this guide:

https://innovastudio.com/contentbuilder-blocks

Note that module files are located in:

assets/modules/

You will see there are some prebuilt modules in the folder (please see the above guide for details).

Your Own Snippets

You can create or add your own snippets by customizing the snippet file:

assets/minimalist-blocks/content.js

The snippet file is JSON formatted.

Here is an example of a snippet definition in the snippet file:

```
{
    'thumbnail': 'preview/basic-01.png',
    'category': '120',
    'html':
        '<div class="row">' +
            '<div class="column full">' +
                '<h1>My Heading Text</h1>' +
            '</div>' +
        '</div>'
}
```

A snippet needs to have a thumbnail, category number and HTML content.

Thumbnails are created & stored in the **preview/** folder:

assets/minimalist-blocks/preview/

Snippet categories you can use:

- 120: Basic
- 118: Article
- 101: Headline
- 119: Buttons
- 102: Photos
- 103: Profile
- 116: Contact
- 104: Products
- 105: Features
- 106: Process
- 107: Pricing
- 108: Skills
- 109: Achievements
- 110: Quotes
- 111: Partners
- 112: As Featured On
- 113: Page Not Found
- 114: Coming Soon
- 115: Help, FAQ

The HTML content for the snippets needs to be formatted in a simple grid. Here are some examples:

Single Column:

```
<div class="row">
    <div class="column full">
    ...
    </div>
</div>
```

Two Columns:

```
<div class="row">
    <div class="column half">
        ...
    </div>
    <div class="column half">
        ...
    </div>
</div>
```

Three Columns:

```
<div class="row">
    <div class="column third">
        ...
    </div>
    <div class="column third">
        ...
    </div>
    <div class="column third">
        ...
    </div>
</div>
```

If you are using a css framework, such as Bootstrap or Foundation, this simple grid will be automatically converted to the framework grid you're using.

Four Columns:

```
<div class="row">
    <div class="fourth third">
        ...
    </div>
    <div class="fourth third">
        ...
    </div>
    <div class="fourth third">
        ...
    </div>
    <div class="fourth third">
        ...
    </div>
</div>
```

Five Columns:

```
<div class="row">
    <div class="column fifth">
        ...
    </div>
    <div class="column fifth">
        ...
    </div>
    <div class="column fifth">
        ...
    </div>
    <div class="column fifth">
        ...
    </div>
    <div class="column fifth">
        ...
    </div>
</div>
```

Six Columns:

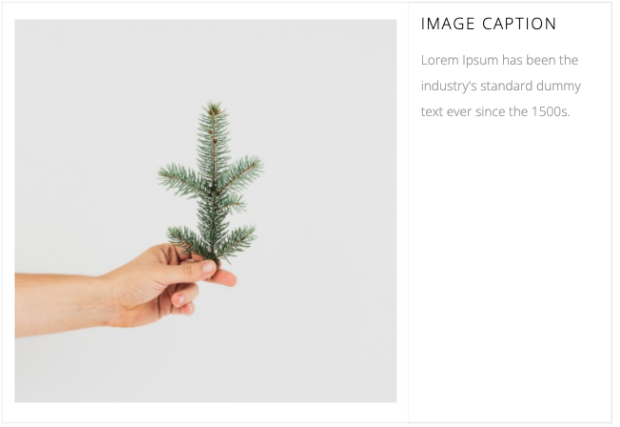
```
<div class="row">
    <div class="column sixth">
        ...
    </div>
    <div class="column sixth">
        ...
    </div>
    <div class="column sixth">
        ...
    </div>
    <div class="column sixth">
        ...
    </div>
    <div class="column sixth">
        ...
    </div>
    <div class="column sixth">
        ...
    </div>
</div>
```

ContentBuilder.js has default maximum number of columns is set to 4. You can increase this limit by setting:

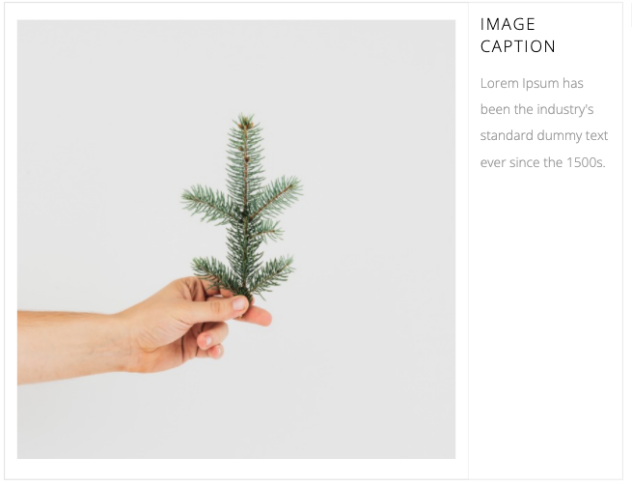
```
const builder = new ContentBuilder({
    container: '.container',
    maxColumns: 5
});
```

Other layouts you can use to create your own your snippets:

```
<div class="row">
    <div class="column two-third">
        ...
    </div>
    <div class="column third">
        ...
    </div>
</div>
```

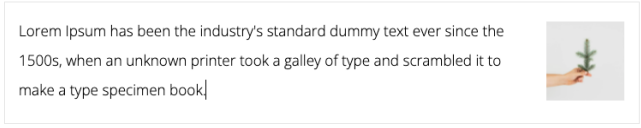



```
<div class="row">
    <div class="column two-fourth">
        ...
    </div>
    <div class="column fourth">
        ...
    </div>
</div>
```



```
<div class="row">
    <div class="column two-fifth">
        ...
    </div>
    <div class="column fifth">
        ...
    </div>
</div>
```


```
<div class="row">
    <div class="column two-sixth">
        ...
    </div>
    <div class="column sixth">
        ...
    </div>
</div>
```



You can also add some behavior in your HTML snippets:

To make an image not replaceable, add the **data-fixed** attribute to the element, for example:

```
<img src=".." data-fixed />
```

To make a column not editable, add the **data-noedit** attribute to the column, for example:

```
<div class="row">
    <div class="column full" data-noedit>
        ...
    </div>
</div>
```

To make a column not editable and cannot be deleted, moved or duplicated, add the **data-protected** attribute to the column, for example:

```
<div class="row">
    <div class="column full" data-protected>
        ...
    </div>
</div>
```

CSS Framework Support

ContentBuilder.js can be used with popular css frameworks, such as Bootstrap, Foundation and more. There are 2 types of configuration:

1. If the framework has 12 columns grid system

Specify the framework classes using the **row** & **cols** parameters.

Example:

Bootstrap:

```
const builder = new ContentBuilder({
    container: '.container',
    row: 'row',
    cols: ['col-md-1', 'col-md-2', 'col-md-3', 'col-md-4', 'col-md-5', 'col-md-6',
        'col-md-7', 'col-md-8', 'col-md-9', 'col-md-10', 'col-md-11', 'col-md-12']
});
```

Foundation:

```
const builder = new ContentBuilder({
    container: '.container',
    row: 'row',
    cols: ['large-1 columns', 'large-2 columns', 'large-3 columns', 'large-4 columns',
        'large-5 columns', 'large-6 columns', 'large-7 columns', 'large-8 columns',
        'large-9 columns', 'large-10 columns', 'large-11 columns', 'large-12 columns']
});
```

Tailwind:

```
const builder = new ContentBuilder({
    container: '.container',
    row: 'flex flex-col md:flex-row',
    cols: [
        'w-full md:w-1/12 px-4',
        'w-full md:w-2/12 px-4',
        'w-full md:w-3/12 px-4',
        'w-full md:w-4/12 px-4',
        'w-full md:w-5/12 px-4',
        'w-full md:w-6/12 px-4',
        'w-full md:w-7/12 px-4',
        'w-full md:w-8/12 px-4',
        'w-full md:w-9/12 px-4',
        'w-full md:w-10/12 px-4',
        'w-full md:w-11/12 px-4',
        'w-full px-4'],
});
```

2. If the framework has grid system in which the column size increment is not constant

Specify two additional parameters: **colequal** & **colsizes**

- **colequal**: list of all class combinations that have same width
- **colsizes**: list of all class combinations in increment order

Example:

UIKit Framework:

```
const builder = new ContentBuilder({
    container: '.container',
    row: 'uk-grid',
    cols: ['uk-width-1-6@m', 'uk-width-1-5@m', 'uk-width-1-4@m', 'uk-width-1-3@m', 'uk-width-2-5@m',
        'uk-width-1-2@m', 'uk-width-3-5@m', 'uk-width-2-3@m', 'uk-width-3-4@m', 'uk-width-4-5@m',
        'uk-width-5-6@m', 'uk-width-1-1@m'],

    //the following parameters are needed for grid system in which the column size increment is not constant.
    colequal: [
        ['uk-width-1-4@m', 'uk-width-1-4@m', 'uk-width-1-4@m', 'uk-width-1-4@m'],
        ['uk-width-1-3@m', 'uk-width-1-3@m', 'uk-width-1-3@m'],
        ['uk-width-1-2@m', 'uk-width-1-2@m']
    ],
    colsizes: [
        [   //increment for 3 columns
            ['uk-width-1-5@m', 'uk-width-2-5@m', 'uk-width-2-5@m'],
            ['uk-width-1-3@m', 'uk-width-1-3@m', 'uk-width-1-3@m'],
            ['uk-width-2-5@m', 'uk-width-2-5@m', 'uk-width-1-5@m'],
            ['uk-width-1-2@m', 'uk-width-1-4@m', 'uk-width-1-4@m'],
            ['uk-width-3-5@m', 'uk-width-1-5@m', 'uk-width-1-5@m']
        ],
        [   //increment for 2 columns
            ['uk-width-1-6@m', 'uk-width-5-6@m'],
            ['uk-width-1-5@m', 'uk-width-4-5@m'],
            ['uk-width-1-4@m', 'uk-width-3-4@m'],
            ['uk-width-1-3@m', 'uk-width-2-3@m'],
            ['uk-width-2-5@m', 'uk-width-3-5@m'],
            ['uk-width-1-2@m', 'uk-width-1-2@m'],
```

Note:

- With the configuration explained above, all snippets/blocks are automatically converted into the framework's grid system. So there is no modification needed on the snippet file when used in various frameworks.
- ContentBuilder.js may not always work on certain frameworks.
- Here are some examples on using various css frameworks:
    - example1.html (without framework / use built in basic css)
    - example1-bootstrap.html (Bootstrap Example)
    - example1-bulma.html (Bulma Example)
    - example1-foundation.html (Foundation Example)
    - example1-foundationxy.html (Foundation XY Grid Example)
    - example1-material.html (Material Design Lite Example)
    - example1-materializecss.html (Materialize Example)
    - example1-milligram.html (Milligram Example)
    - example1-minicss.html (Mini.css Example)
    - example1-mustardui.html (Mustard UI Example)
    - example1-primer.html (Primer Example)
    - example1-purecss.html (Pure Css Example)
    - example1-skeleton.html (Skeleton Example)
    - example1-spectre.html (Spectre.css Example)
    - example1-tailwind (Tailwind Example)
    - example1-uikit.html (UIKit Example)
    - example1-w3css.html (W3.CSS Example)

Multiple Editable Area Support

To implement multiple editable area, Use the same initialization.

```
<div class="section">
    <div class="container area1">
        ...
    </div>
</div>
<div class="section">
    <div class="container area2">
        ...
    </div>
</div>
```

```
const builder = new ContentBuilder({
    container: '.container'
});
```

To get the HTML content:

```
let html1 = builder.html(document.querySelector('.area1')); //Get 1st area content

let html2 = builder.html(document.querySelector('.area2')); //Get 2nd area content
```

To try, please open **example3.html**.

Programmatically Add Editable Area

If your application dynamically adds a container to edit, you don't need to re-init the ContentBuilder.js.

Just run **applyBehavior()** method:

```
builder.applyBehavior();
```

This will make the new container editable.

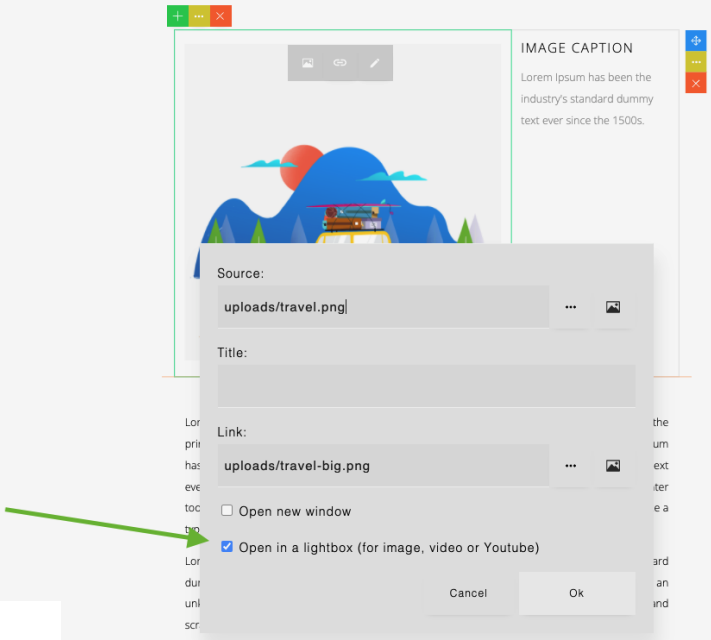
Lightbox Support

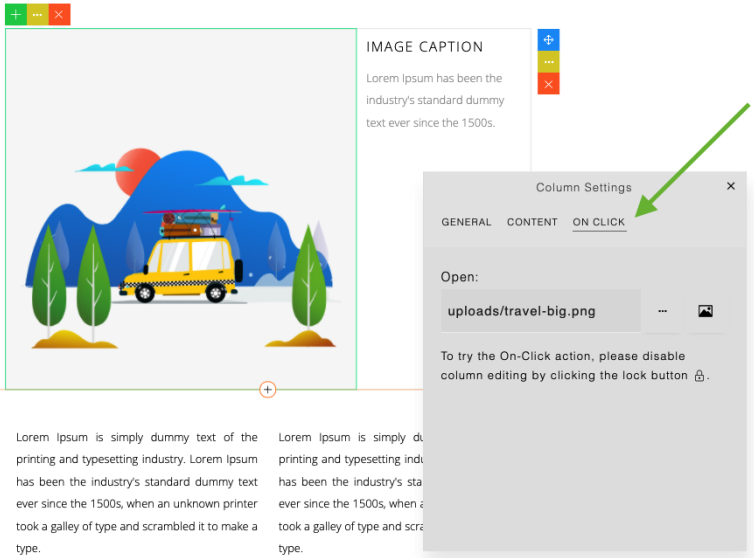
With Lightbox support, you can click a link or column to open an image, video (.mp4) or Youtube in an animated modal window (a lightbox).

To enable this feature:

```
const builder = new ContentBuilder({
    container: '.container',
    useLightbox: true // default: false
});
```

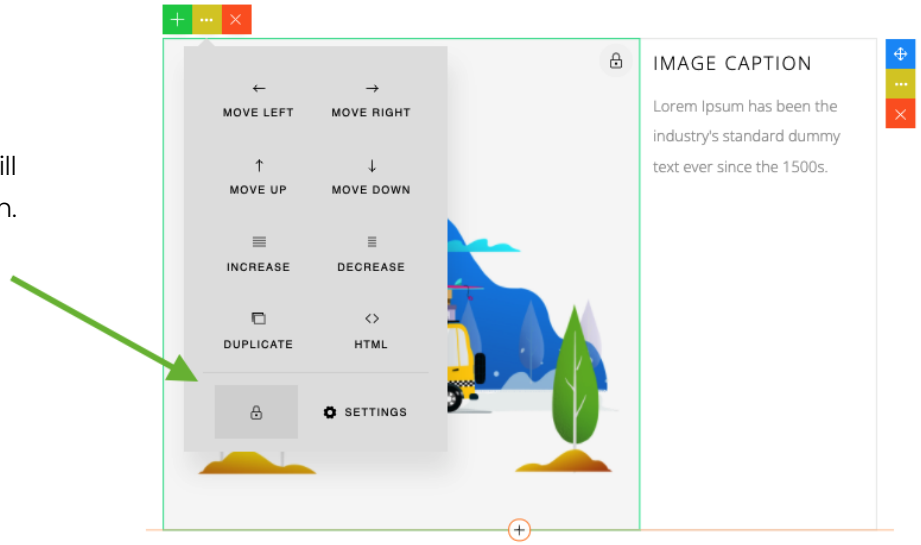
If lightbox support is enabled, a checkbox to open the image or video link in a lightbox will be displayed in the Image and link dialog.



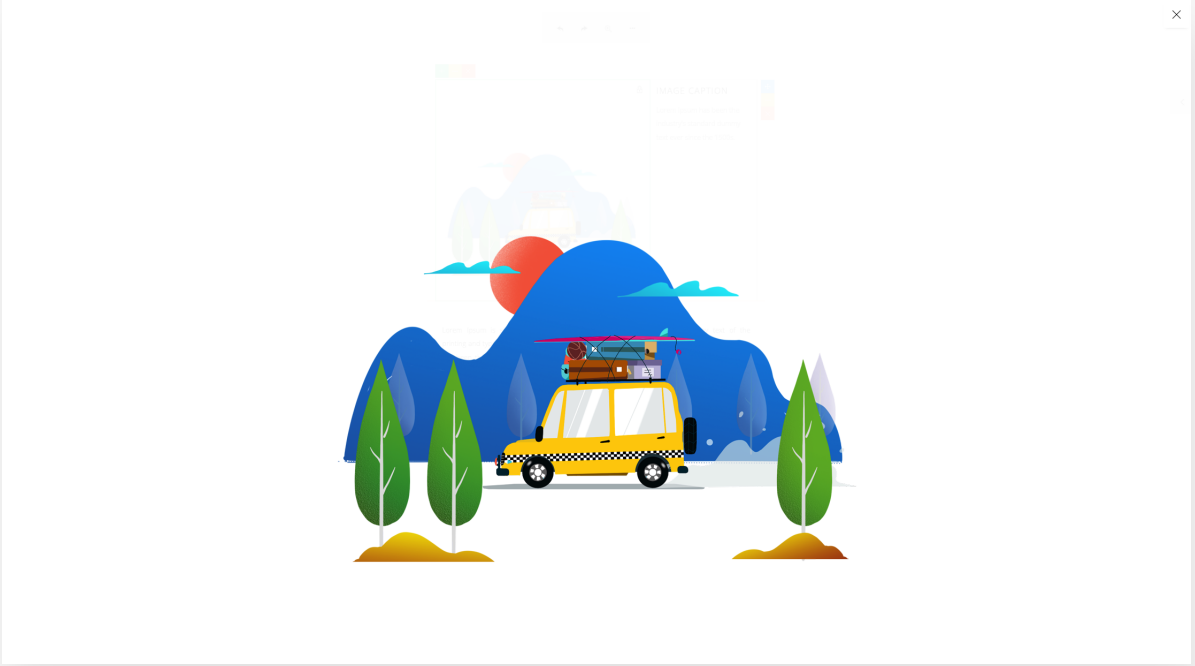


Also in the Column Settings dialog, the 'On Click' tab will be visible.

To try the On-Click action, you will need to lock the column.



Then, when you click the link or column, the lightbox will be opened (a locked column will disable the editing).



In Production

In production (for viewing content), you will need to include a Lightbox script as follow:

The css:

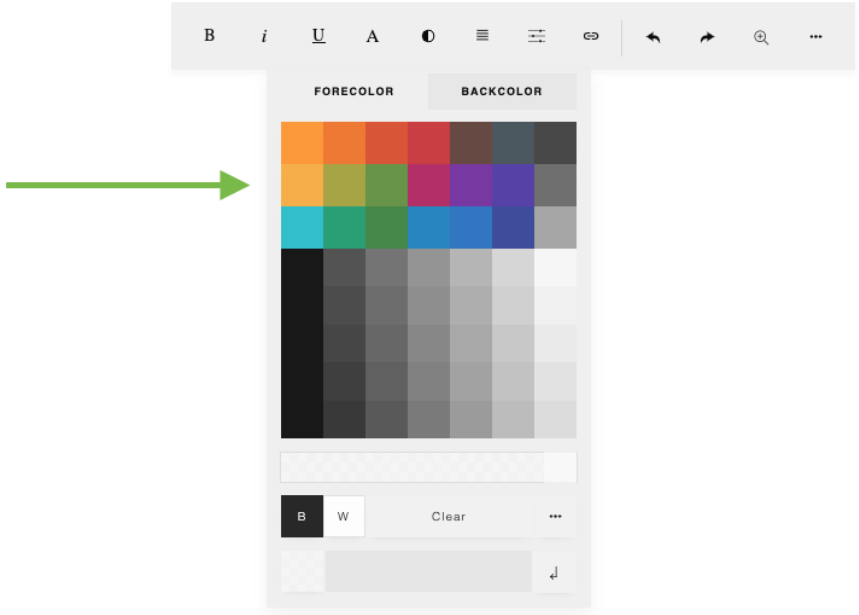
```
<link href="assets/scripts/lightbox/lightbox.css" rel="stylesheet" type="text/css" />
```

And the js:

```
<script src="assets/scripts/lightbox/lightbox.js"></script>
<script>
    lightbox.init();
</script>
```

Color Picker

The color picker shows some default color list.

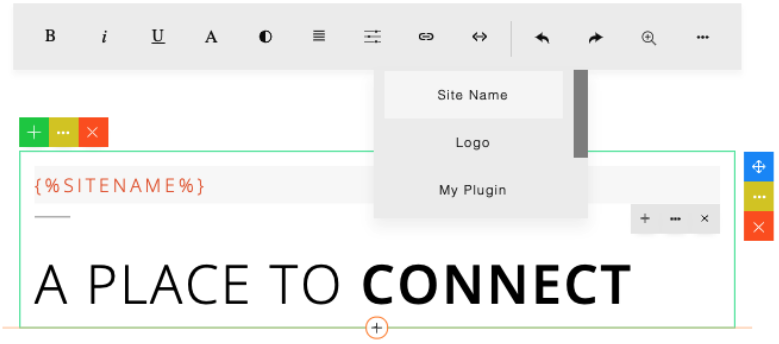


To configure the colors:

```
const builder = new ContentBuilder({
    container: '.container',
    colors: ["#ff8f00", "#ef6c00", "#d84315", "#c62828", "#58362f", "#37474f", "#353535",
        "#f9a825", "#9e9d24", "#558b2f", "#ad1457", "#6a1b9a", "#4527a0", "#616161",
        "#00b8c9", "#009666", "#2e7d32", "#0277bd", "#1565c0", "#283593", "#9e9e9e"],
});
```

Custom Tags

Custom tags are commonly used in a CMS for adding dynamic element within the content by replacing the tags in production (not during editing). Here is an example:



To configure this feature:

```
const builder = new ContentBuilder({
    container: '.container',
    customTags: [["Site Name", "{%SITENAME%}"],
        ["Logo", "{%LOGO%}"],
        ["My Plugin", "{%MY_PLUGIN%}"],
});
```

Modal Animation

Some modals open with animation. For example, when deleting a row, a confirmation modal shows with zoom animation on the content. To enable or disable this feature:

```
const builder = new ContentBuilder({
    container: '.container',
    animateModal: false, // default: true
});
```

Custom Value

You can pass any custom value to the server. In a CMS application, you can pass, for example, a user id or session id, etc.

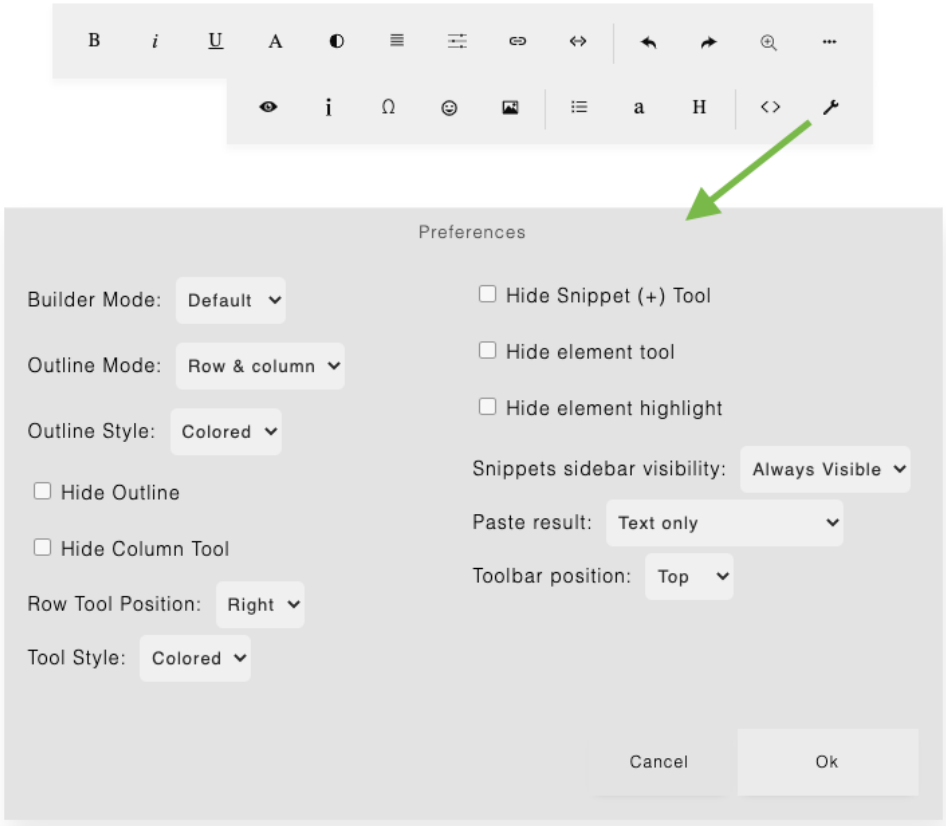
The value will be passed when an image is submitted to the server using the Form method. Note that in the AJAX post method, you may not need this feature as you can post any data during the AJAX post. On the server, the image handler can use this value to decide where to save the image for each user. This is more towards customizing your application.

To specify a custom value:

```
const builder = new ContentBuilder({
    container: '.container',
    customval: 200 // or any value
});
```

Preferences

The preferences dialog allows you to choose your editing preferences. You can hide unwanted tools, choose the toolbar placement or even make the interface clean to write without distraction.



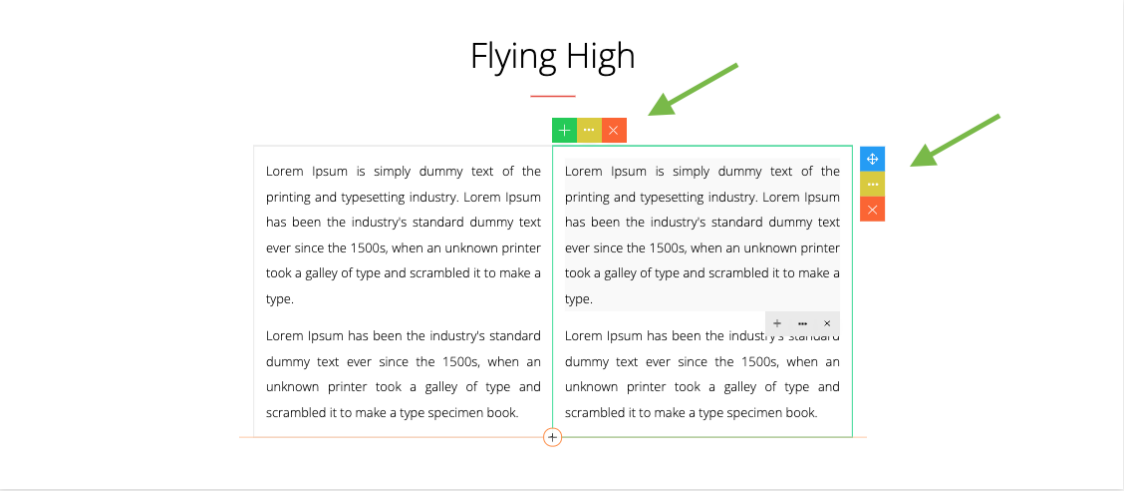
Preference options are explained below.

Builder Mode

You can choose: Default, Minimal or Clean.

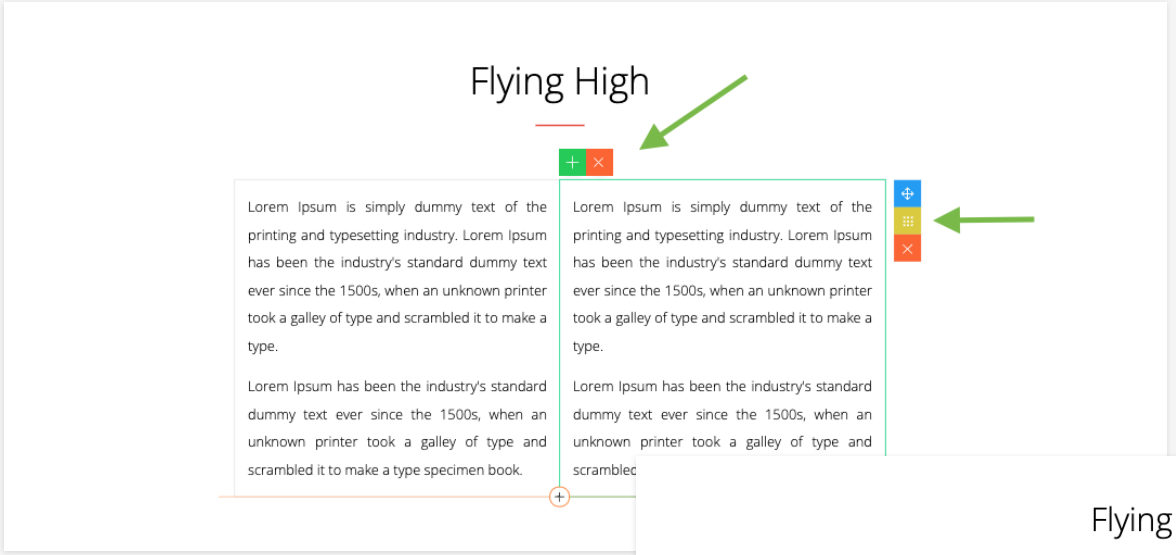
Default

The default mode shows all available buttons on the row & column tool.



Minimal

The minimal mode simplifies the row & column tool.



As seen in the row tool, a new 'Grid' button is shown. Clicking on the button will show a small grid tool.

Clean

This option will further simplify the interface, by showing only the 'Grid' button.

To set the Builder Mode in code:

```
const builder = new ContentBuilder({
    container: '.container',
    builderMode: 'clean'
});
```

Values:

- not set or empty (default)
- minimal
- clean

Outline Mode

You can choose: Row & Column, Row Only.

Row & Column

Shows an outline on active row and column.

Row Only

Shows an outline on active row only.

To set the Outline Mode in code:

```
const builder = new ContentBuilder({
    container: '.container',
    outlineMode: 'row'
});
```

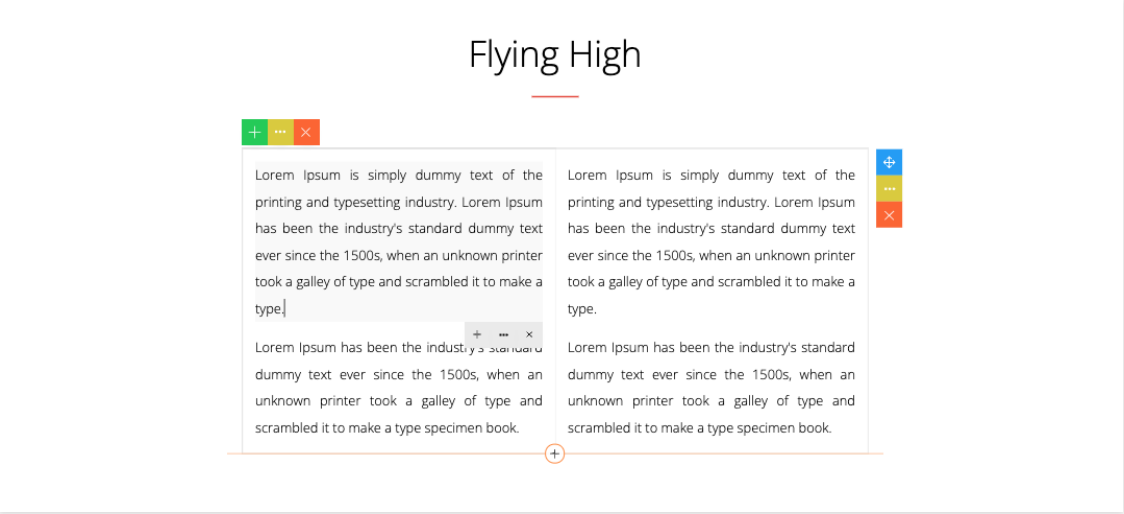
Values:

- not set or empty => Row & Column (default)
- row => Row Only

Outline Style

You can choose: Colored, Gray.

Here the outline is set to gray.



To set the Outline Style in code:

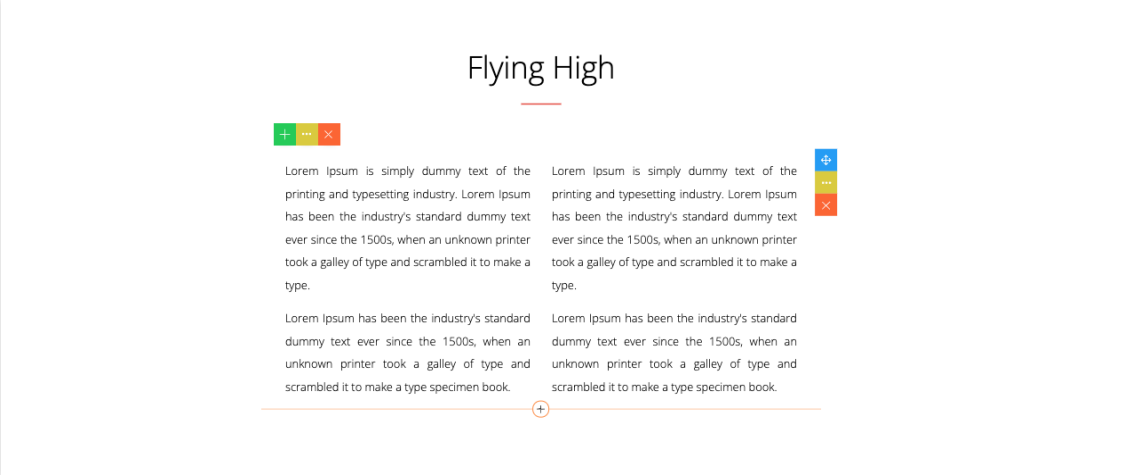
```
const builder = new ContentBuilder({
    container: '.container',
    outlineStyle: 'grayoutline'
});
```

Values:

- not set or empty => Colored (default)
- grayoutline => Gray

Hide Outline

If checked, there will be no outline on active row or column.



To hide the Outline in code:

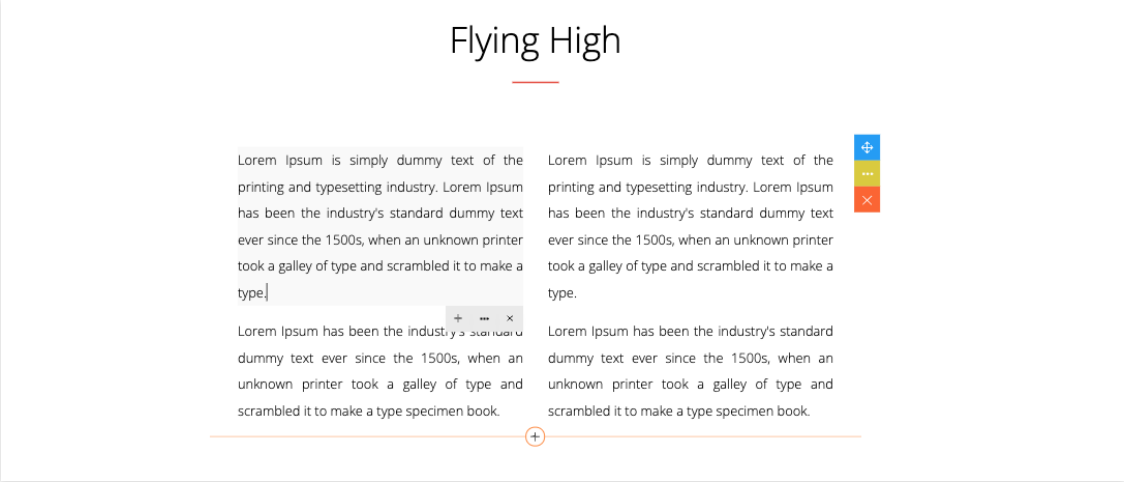
```
const builder = new ContentBuilder({
    container: '.container',
    rowcolOutline: false // outline will be hidden
});
```

Values:

- true => Visible (default)
- false => Hidden

Hide Column Tool

If checked, column tool will not be visible on active column.



To hide the Column Tool in code:

```
const builder = new ContentBuilder({
    container: '.container',
    columnTool: false // column tool will be hidden
});
```

Values:

- true => Visible (default)
- false => Hidden

Row Tool Position

You can choose: Left, Right.

Here the position is set Left.

To set the Row Tool Position in code:

```
const builder = new ContentBuilder({
    container: '.container',
    rowTool: 'left' // positioned on left
});
```

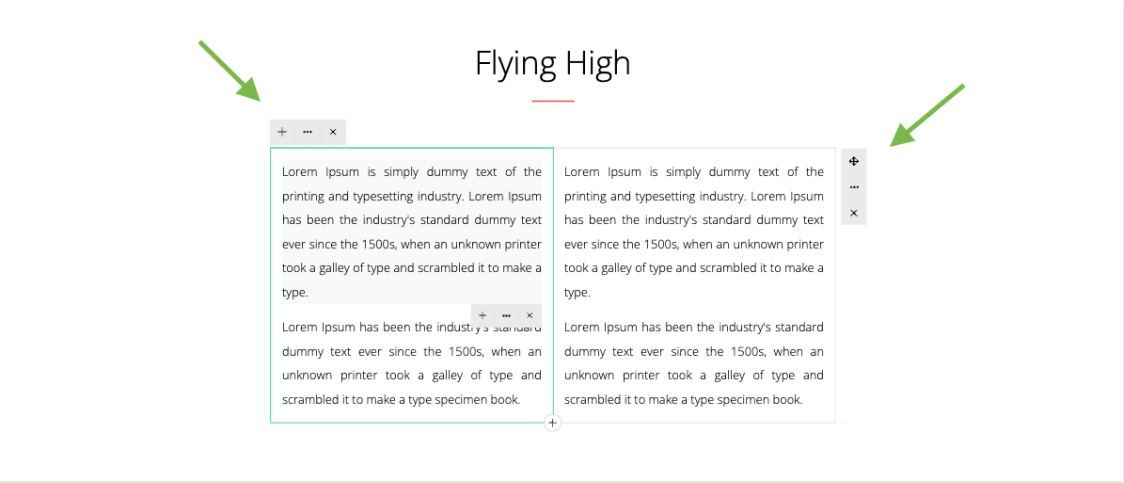
Values:

- right (default)
- left

Tool Style

You can choose: Colored, Mono.

Here the tool style is set to Mono.



To set the Tool Style in code:

```
const builder = new ContentBuilder({
    container: '.container',
    toolStyle: 'gray' // = Gray or Mono
});
```

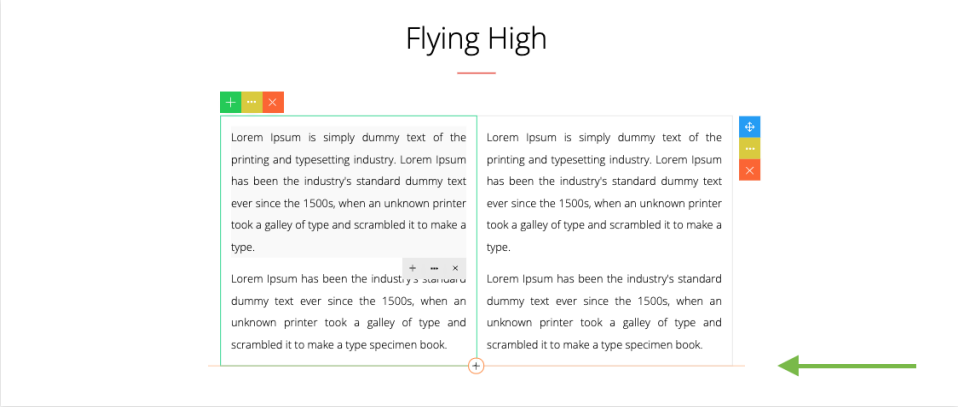
Values:

- not set or empty => Colored (default)
- gray => Gray or Mono

Hide Snippet (+) Tool

If checked, the Add (+) Snippet button will be hidden.

Here the Add (+) Snippet button is visible.



And here the Add (+) Snippet button is hidden.

To hide the Snippet (+) Tool in code:

```
const builder = new ContentBuilder({
    container: '.container',
    snippetAddTool: false // Snippet (+) tool will be hidden
});
```

Values:

- true => Visible (default)
- false => Hidden

Hide Element Tool

If checked, the element tool will be hidden.

Here the element tool is visible.

And here the element tool is hidden.

To hide the Element Tool in code:

```
const builder = new ContentBuilder({
    container: '.container',
    elementTool: false // Element tool will be hidden
});
```

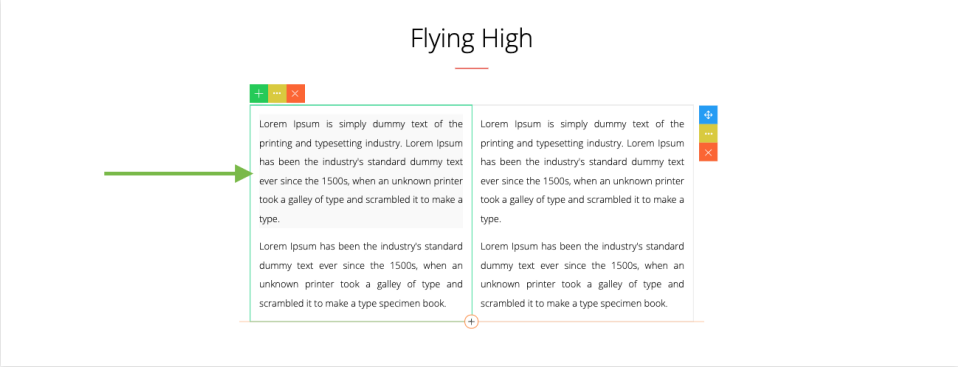
Values:

- true => Visible (default)
- false => Hidden

Hide Element Highlight

If checked, there will be no active element highlight.

Here the element highlight is visible.



And here the element highlight is not visible.

To hide the Element Highlight in code:

```
const builder = new ContentBuilder({
    container: '.container',
    elementHighlight: false // Element highlight will be hidden
});
```

Values:

- true => Visible (default)
- false => Hidden

Snippet Sidebar Visibility

You can choose: Auto, Always Visible.

Normally, the snippet sidebar will be automatically closed during editing. If set to 'Always Visible', the snippet sidebar will stay visible.

To set the Snippet Sidebar Visibility in code:

```
const builder = new ContentBuilder({
    container: '.container',
    snippetDisplay: 'visible' // stay visible
});
```

Values:

- auto => Auto close (default)
- visible => Stay visible

Paste Result

You can choose:

- Text Only
- HTML (Without Style)
- HTML (With Style)

This is a behavior you can choose when you paste copied content from another source, for example, from a Word document.

To set the Paste Result in code:

```
const builder = new ContentBuilder({
    container: '.container',
    paste: 'text'
});
```

Values:

- text (default)
- html
- html-without-styles

Toolbar Position

You can choose: Top, Left, or Right.

If the position is set to Left, the toolbar will be shown on the left.

If the position is set to Right, the toolbar will be shown on the right.

To set the Toolbar Position in code:

```
const builder = new ContentBuilder({
    container: '.container',
    toolbar: 'left'
});
```

Values:

- top (default)
- left
- right

Programmatically Open Preference Dialog

Preference dialog can be opened programmatically using:

```
builder.viewConfig();
```

Clear Saved Preferences

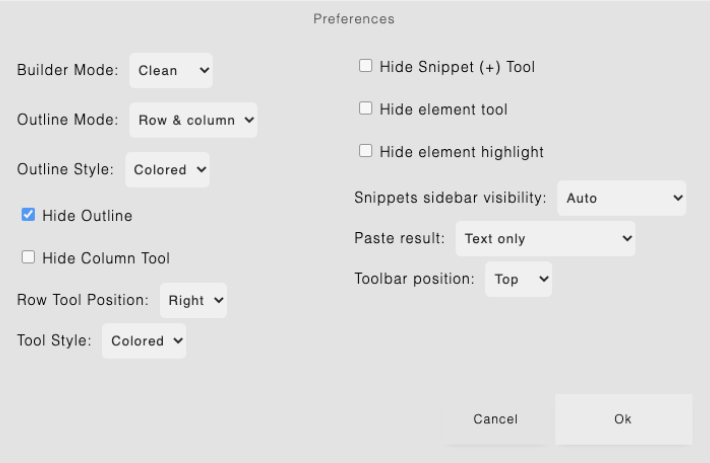
Normaly, when you click Ok on the Preferences dialog, your chosen settings will be automatically saved (on browser local storage). So the next time you open the builder, you can still work with your chosen preferences.

To force the builder to not use the saved preferences, you can use:

```
const builder = new ContentBuilder({
    container: '.container',
    clearPreferences: true,
});
```

Example: Making the builder interface clean

As an example, here are the settings you can choose to make the builder clean.



In this example, the Builder Mode is set to 'Clean' and the Hide Outline is checked.

Here is the result.

Themes

To enable theme selection (with complete list of themes):

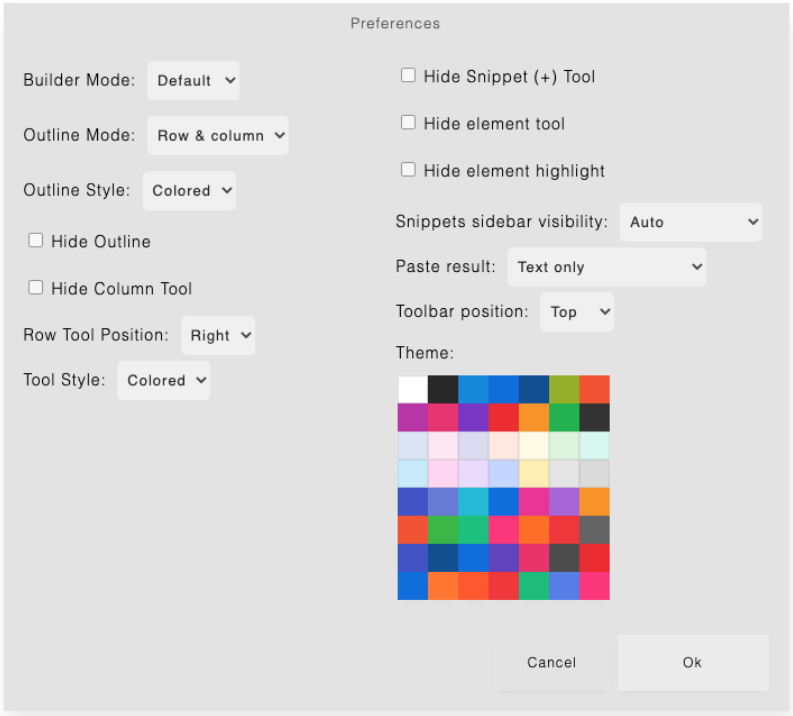
```
const builder = new ContentBuilder({
    container: '.container',
    themes: [
        ['#ffffff','',''],
        ['#282828','dark','contentbuilder/themes/dark.css'],
        ['#0088dc','colored','contentbuilder/themes/colored-blue.css'],
        ['#006add','colored','contentbuilder/themes/colored-blue6.css'],
        ['#0a4d92','colored','contentbuilder/themes/colored-darkblue.css'],
        ['#96af16','colored','contentbuilder/themes/colored-green.css'],
        ['#f3522b','colored','contentbuilder/themes/colored-orange.css'],

        ['#b92ea6','colored','contentbuilder/themes/colored-magenta.css'],
        ['#e73171','colored','contentbuilder/themes/colored-pink.css'],
        ['#782ec5','colored','contentbuilder/themes/colored-purple.css'],
        ['#ed2828','colored','contentbuilder/themes/colored-red.css'],
        ['#f9930f','colored','contentbuilder/themes/colored-yellow.css'],
        ['#13b34b','colored','contentbuilder/themes/colored-green4.css'],
        ['#333333','colored-dark','contentbuilder/themes/colored-dark.css'],

        ['#dbe5f5','light','contentbuilder/themes/light-blue.css'],
        ['#fbe6f2','light','contentbuilder/themes/light-pink.css'],
        ['#dcdaf3','light','contentbuilder/themes/light-purple.css'],
        ['#ffe9e0','light','contentbuilder/themes/light-red.css'],
        ['#fffae5','light','contentbuilder/themes/light-yellow.css'],
        ['#ddf3dc','light','contentbuilder/themes/light-green.css'],
        ['#c7ebfd','light','contentbuilder/themes/light-blue2.css'],
```

Each theme definition consists of:

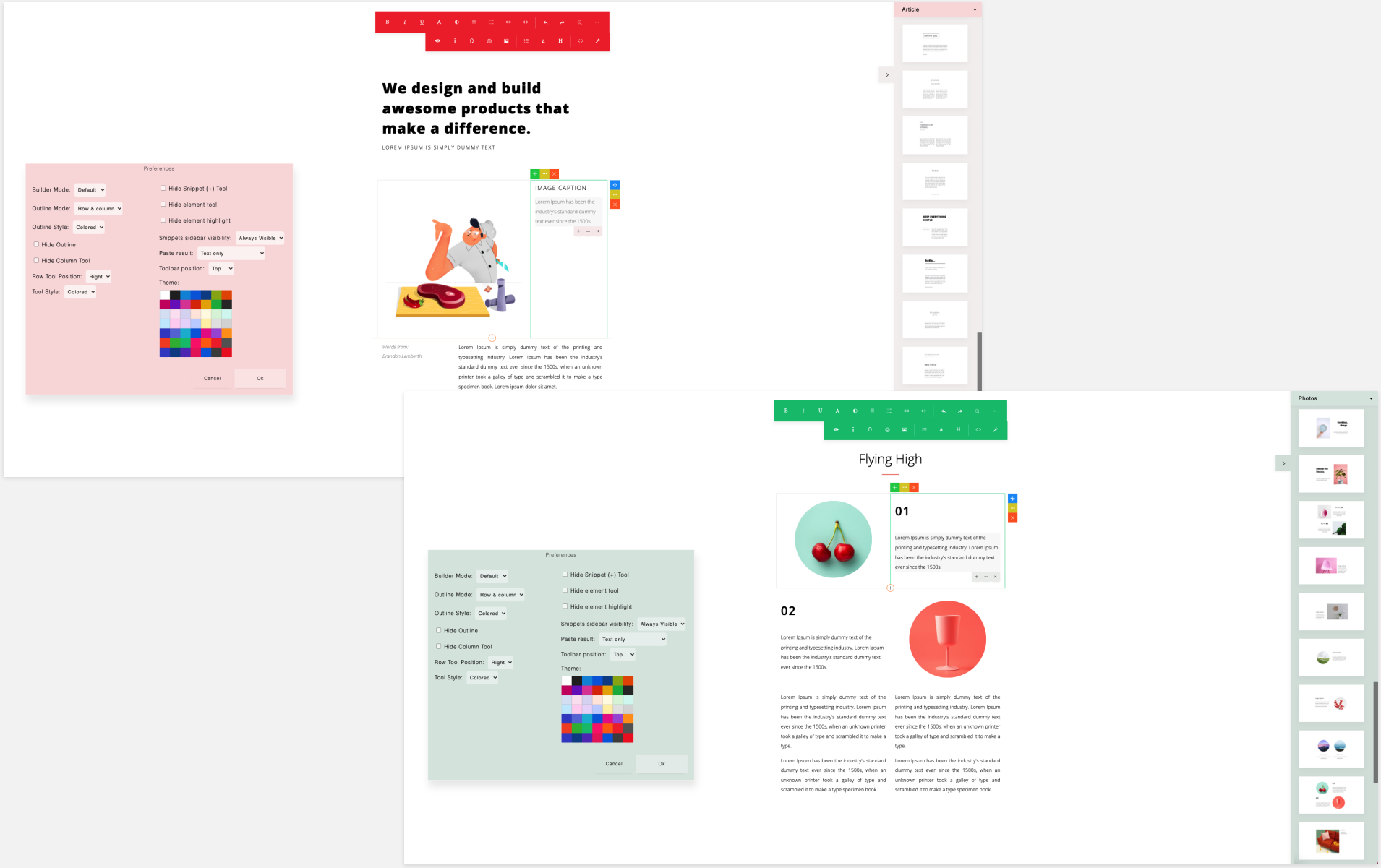
- color preview (for the selection)
- class name (needed by the css)
- css file

```
['#3f4ec9','colored','contentbuilder/themes/colored-blue2.css'],
['#6779d9','colored','contentbuilder/themes/colored-blue4.css'],
['#10b9d7','colored','contentbuilder/themes/colored-blue3.css'],
['#006add','colored','contentbuilder/themes/colored-blue5.css'],
['#e92f94','colored','contentbuilder/themes/colored-pink3.css'],
['#a761d9','colored','contentbuilder/themes/colored-purple2.css'],
['#f9930f','colored','contentbuilder/themes/colored-yellow2.css'],

['#f3522b','colored','contentbuilder/themes/colored-red3.css'],
['#36b741','colored','contentbuilder/themes/colored-green2.css'],
['#00c17c','colored','contentbuilder/themes/colored-green3.css'],
['#fb3279','colored','contentbuilder/themes/colored-pink2.css'],
['#ff6d13','colored','contentbuilder/themes/colored-orange2.css'],
['#f13535','colored','contentbuilder/themes/colored-red2.css'],
['#646464','colored','contentbuilder/themes/colored-gray.css'],

['#3f4ec9','dark','contentbuilder/themes/dark-blue.css'],
['#0b4d92','dark','contentbuilder/themes/dark-blue2.css'],
['#006add','dark','contentbuilder/themes/dark-blue3.css'],
['#5f3ebf','dark','contentbuilder/themes/dark-purple.css'],
['#e92f69','dark','contentbuilder/themes/dark-pink.css'],
['#4c4c4c','dark','contentbuilder/themes/dark-gray.css'],
['#ed2828','dark','contentbuilder/themes/dark-red.css'],

['#006add','colored','contentbuilder/themes/colored-blue8.css'],
['#ff7723','colored','contentbuilder/themes/colored-orange3.css'],
['#ff5722','colored','contentbuilder/themes/colored-red5.css'],
['#f13535','colored','contentbuilder/themes/colored-red4.css'],
```

When enabled, the theme selection will be visible on the Preferences dialog.



Example themes:



To load a specific theme programmatically, you can use:

```
builder.setUIColor('colored','contentbuilder/contentbuilder-red.css');
```

For light theme:

```
builder.setUIColor('');
```

For dark theme:

```
builder.setUIColor('dark','contentbuilder/contentbuilder-dark.css');
```

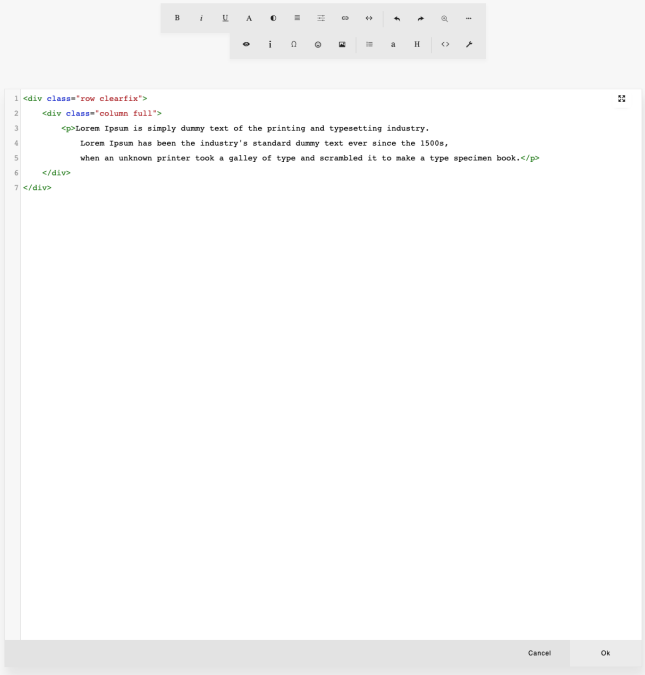
HTML Code Editing Support

You can edit the HTML code of your content with or without syntax highlighting.

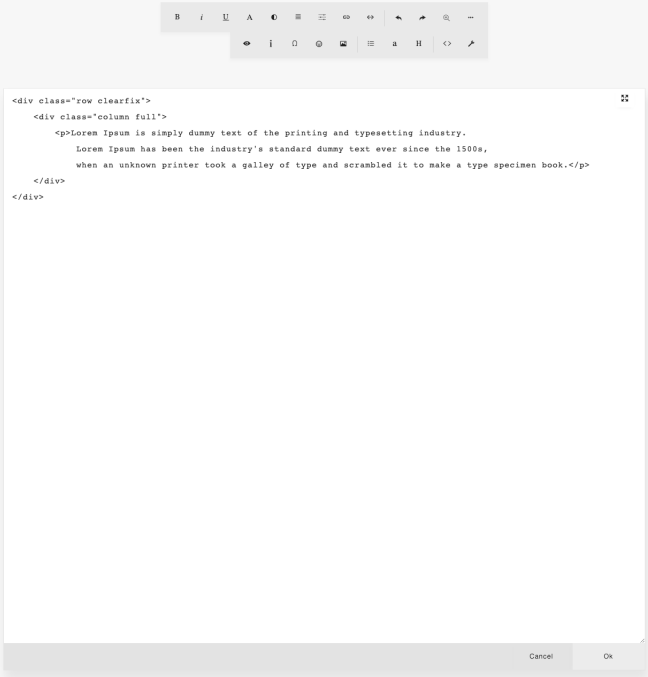
To configure:

```
const builder = new ContentBuilder({
    container: '.container',
    htmlSyntaxHighlighting: true,
});
```

With syntax highlighting:



Without syntax highlighting:



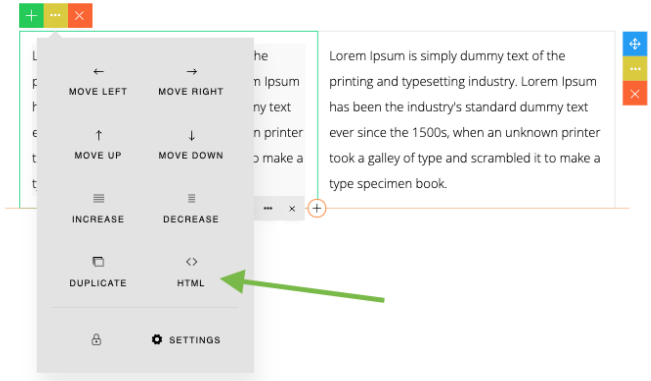
Programmatically Open HTML Code Editor

To open HTML code editor programmatically:

```
builder.viewHtml();
```

Disable Column HTML Code Editor

The column HTML editor button is shown on the column tool popup. If disabled, the button will not be visible.

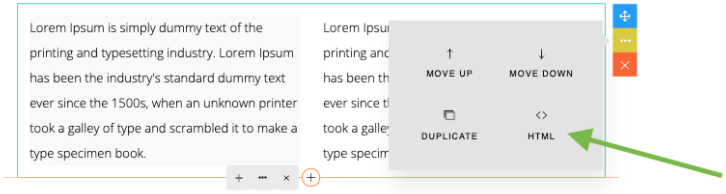


To disable:

```
const builder = new ContentBuilder({
    container: '.container',
    columnHtmlEditor: false,
});
```

Disable Row HTML Code Editor

The row HTML editor button is shown on the row tool popup. If disabled, the button will not be visible.



To disable:

```
const builder = new ContentBuilder({
    container: '.container',
    rowHtmlEditor: false,
});
```

Loading HTML Content Programmatically

To load HTML content programmatically, you can pass your HTML content in the **loadHtml()** method:

```
builder.loadHtml(html);
```

Paste HTML Content Programmatically

To paste HTML content programmatically, you can pass your HTML content in the **pasteHtmlAtCaret()** method:

```
builder.pasteHtmlAtCaret(html);
```

You will need to place the cursor on the content where you want to paste the HTML.

Insert Custom HTML Snippet Programmatically

To insert custom HTML snippet programmatically, you can pass your HTML snippet in the **addSnippet()** method:

```
builder.addSnippet(`
    <div class="row">
        <div class="column">
            <h3>Hello World!</h3>
        </div>
    </div>
`);
```

The HTML snippet must follow the grid structure as explained in chapter '**Your Own Snippets**'.

Element Selection

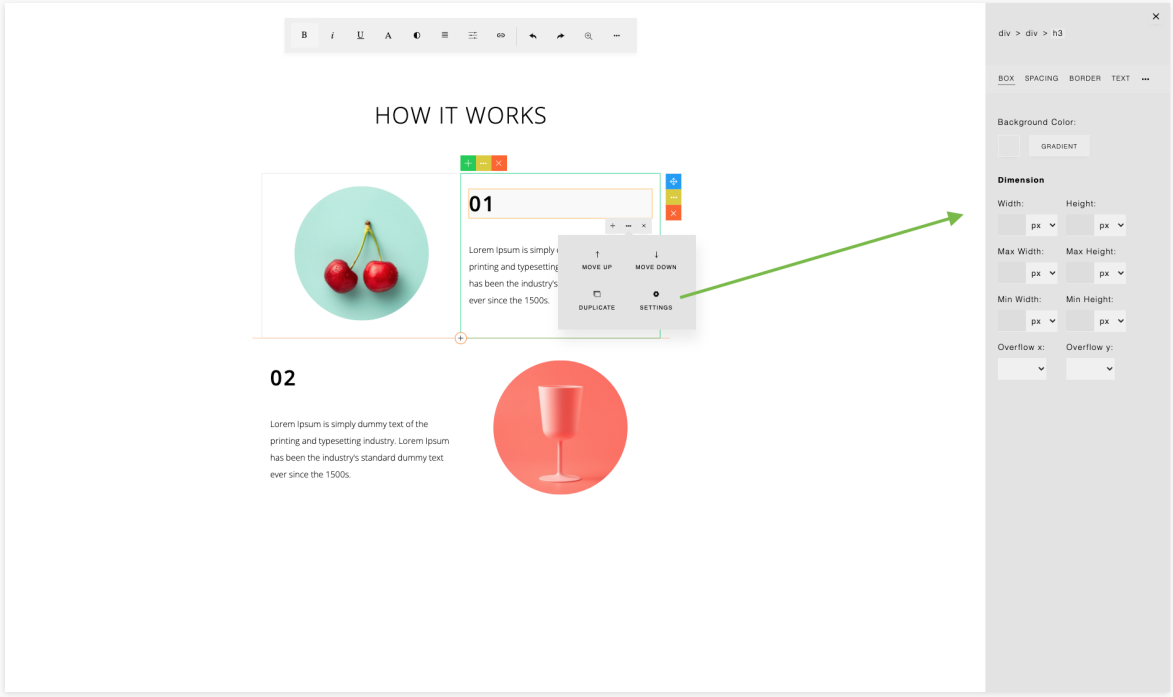
By default, when you press CMD-A or CTRL-A, you will select the current element where cursor is placed (not selecting all elements on the entire column). This behavior is defined by the **elementSelection** parameter:

```
const builder = new ContentBuilder({
    container: '.container',
    elementSelection: true,
});
```

If **elementSelection** is set to false, then the selection will go to the entire column.

Style Panel

ContentBuilder.js provides you with a style panel to format the selected element. Click the Settings button on the element tool popup to open the Style panel.

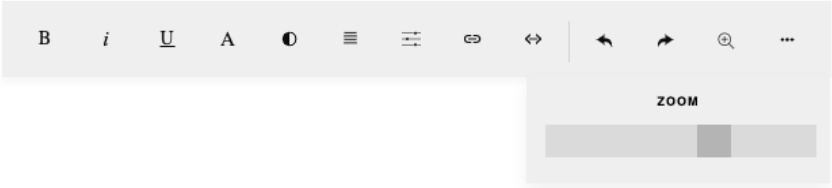


To disable the Style panel button:

```
const builder = new ContentBuilder({
    container: '.container',
    elementEditor: false,
});
```

Zoom Support

You can change the editable area zoom by clicking the zoom icon on the toolbar and use the zoom slider.



To specify the default zoom, use:

```
const builder = new ContentBuilder({
    container: '.container',
    zoom: 0.8, // value: 0.5 to 1
});
```

When you change the zoom size, the value is saved automatically (on browser's local storage). So next time you open the builder, the previously chosen zoom will be used (not the default zoom value).

Some event callbacks you can use when the zoom is changed.

```
const builder = new ContentBuilder({
    container: '.container',
    onZoomStart: function() { ... },
    onZoom: function(scale) { ... },
    onZoomEnd: function(scale) { ... },
});
```

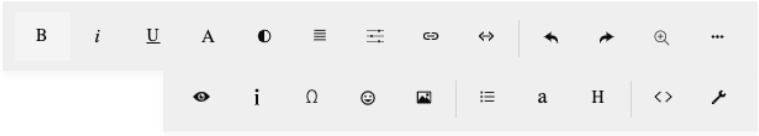
Toolbar

There are two types of toolbars: Main toolbar & Element toolbar.

Main Toolbar

Main toolbar is displayed when a text element is selected.

When you click the ... (more) button, a second row of buttons will be displayed



To configure the main toolbar, use the **buttons** parameter, and to configure the second row of the toolbar, use the **buttonsMore** parameter:

```
const builder = new ContentBuilder({
    container: '.container',
    buttons: ['bold', 'italic', 'underline', 'formatting', 'color', 'align', 'textsettings',
        'createlink', 'tags', '|', 'undo', 'redo', 'zoom', 'more'],
    buttonsMore: ['icon', 'image', '|', 'list', 'font', 'formatpara', '|', 'html', 'preferences'],
});
```

Note: Use '|' for separator.

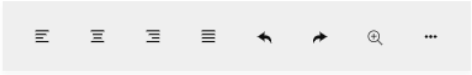
Allowed buttons for the main toolbar:

- addsnippet
- bold
- italic
- underline
- formatting
- color
- removeformat
- align
- textsettings
- createlink
- tags
- undo
- redo
- zoom
- icon
- image
- list
- font
- formatpara
- gridtool
- html
- preferences
- more (cannot be used in second row)
- plugin buttons (such as: preview, wordcount, searchreplace, symbols)

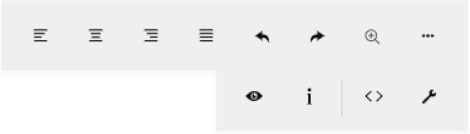
Element Toolbar

Element toolbar is displayed when a non-text element is selected, for example, when you select an image.

The default element toolbar shows the buttons as seen below:



When you click the ... (more) button, a second row of buttons will be displayed:



To configure the element toolbar, use the **elementButtons** parameter, and to configure the second row buttons use the **elementButtonsMore** parameter:

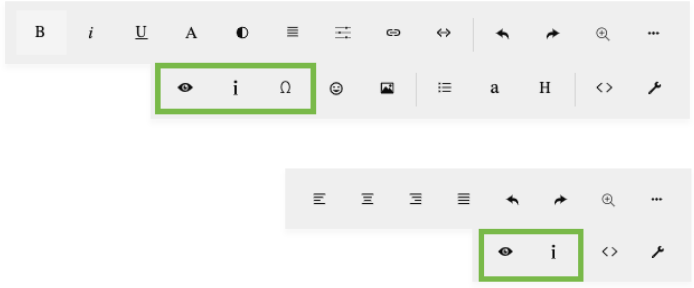
```
const builder = new ContentBuilder({
    container: '.container',
    elementButtons: ['left', 'center', 'right', 'full' , 'undo', 'redo', 'zoom', 'more'],
    elementButtonsMore: ['|', 'html', 'preferences'],
});
```

Allowed buttons for the element toolbar:

- addsnippet
- left
- center
- right
- full
- gridtool
- html
- preferences
- undo
- redo
- zoom
- more (cannot be used in second row)
- plugin buttons (such as: preview, wordcount, searchreplace, symbols)

Plugin Buttons

As you can see on both the main toolbar and the element toolbar, there are some additional buttons that come from installed plugins (on the left side of the second toolbar).



The default installed plugins are:

- preview (adds the **Preview** button to preview the content in various screen sizes)
- wordcount (adds the **Word Count** button to show word count info)
- symbols (adds the **Symbols** button to insert symbols)
- buttoneditor (this plugin doesn't add a button on the toolbar)

If plugin buttons are defined in the toolbar (the plugin name found in **buttons**, **buttonsMore**, **elementButtons** or **elementButtonsMore**), then the plugin buttons will not be automatically added on the left side of the second toolbar.

Please see Plugins section for more info on plugins.

Example: if you don't want to use the "More" button

```
const builder = new ContentBuilder({
    container: '.container',
    buttons: ['bold', 'italic', 'undo', 'redo', 'zoom', 'preview'],
    buttonsMore: [],
    elementButtons: ['undo', 'redo', 'zoom', 'preview'],
    elementButtonsMore: [],
    plugins: [
        { name: 'preview', showInMainToolbar: true, showInElementToolbar: true },
    ],
    pluginPath: 'contentbuilder/',
});
```

As seen on the code, we install only the 'preview' plugin. You will need to define the 'preview' button on the toolbar. Then you set the second row toolbars to empty. Here is the result.

Or, if you are not using any plugins, use:

```
const builder = new ContentBuilder({
    container: '.container',
    buttons: ['bold', 'italic', 'undo', 'redo', 'zoom'],
    buttonsMore: [],
    elementButtons: ['undo', 'redo', 'zoom'],
    elementButtonsMore: [],
    disableConfig: true
});
```

Here is the result.

Icon Toolbar

Another toolbar that is used is the icon toolbar which is displayed when an icon is selected.

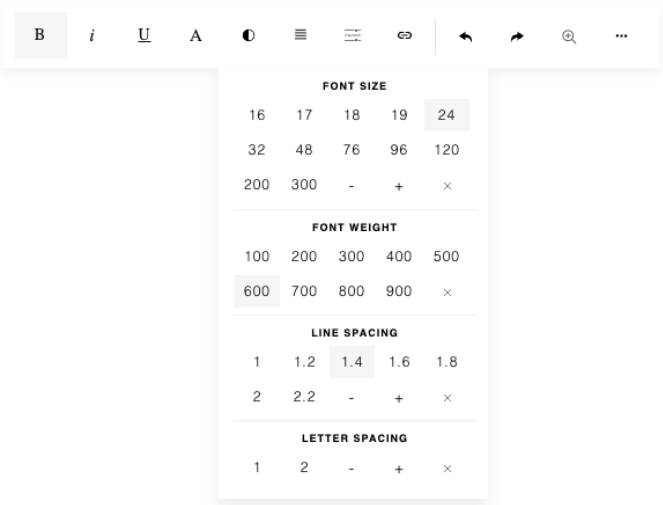
The icon toolbar shows the buttons as seen below:

To configure the icon toolbar, use the **iconButtons** parameter, and to configure the second row buttons, use the **iconButtonsMore** parameter:

```
const builder = new ContentBuilder({
    container: '.container',
    iconButtons: ['icon', 'color','textsettings', 'createLink','|', 'undo', 'redo', 'zoom', 'more'],
    iconButtonsMore: ['|', 'html', 'preferences'],
});
```

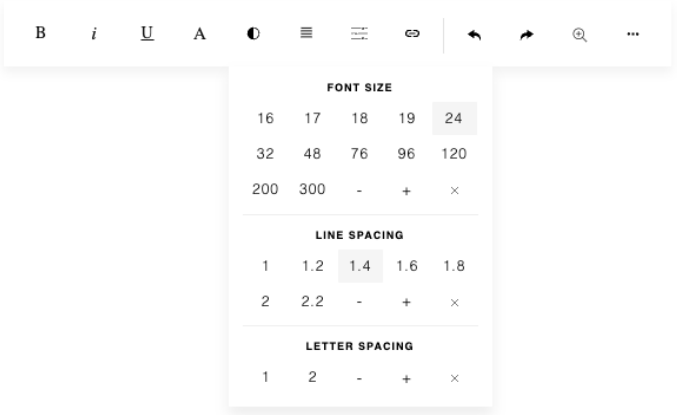
Text Settings

With Text Settings on the toolbar, you can adjust font size, font weight, line spacing (line height) and letter spacing.



If the Bold button on the toolbar is sufficient for you, you can hide the font weight settings by setting the **simpleTextSettings** parameter to true.

```
const builder = new ContentBuilder({
    container: '.container',
    simpleTextSettings: true,
});
```



Undo & Redo

From the toolbar, you can perform an undo or redo operation. You can also perform an undo or redo action programmatically using:

```
builder.undo();
```

or:

```
builder.redo();
```

Events

Some useful parameters to define an event callback function:

onRender

Triggered when content structure is changed, for example, when a new snippet is added. If you monitor your content to apply a certain process or add a certain behavior, you can re-run your process here.

```
const builder = new ContentBuilder({
    container: '.container',
    onRender: function() { ... }
});
```

onChange

Triggered when content is changed.

```
const builder = new ContentBuilder({
    container: '.container',
    onChange: function() { ... }
});
```

onContentClick(event)

Triggered when content is clicked.

```
const builder = new ContentBuilder({
    container: '.container',
    onContentClick: function(event) { ... }
});
```

Destroy

To destroy the builder plugin:

```
builder.destroy();
```

Flexible Path

ContentBuilder.js requires some assets located in the **assets/** folder. For example, there are icons, optional scripts for module, etc.

If you want to change the location of the **assets/** folder, you can change the **assetPath** value:

```
const builder = new ContentBuilder({
    container: '.container',
    assetPath: 'assets/'
});
```

The **assetPath** value is required by the icons and optional scripts for module.

There are some folders inside the **assets/** folder:

assets/fonts/

This folder contains thumbnails for the font selection. You can change the location of this folder by changing the value of the **fontAssetPath** parameter:

```
const builder = new ContentBuilder({
    container: '.container',
    fontAssetPath: 'assets/fonts/'
});
```

assets/minimalist-blocks/

This is the location of the snippet file and its assets. To change the location:

```
const builder = new ContentBuilder({
    container: '.container',
    snippetUrl: 'assets/minimalist-blocks/content.js', // Snippet file
    snippetPath: 'assets/minimalist-blocks/',  // Location of snippets' assets
});
```

You may also need to use the **snippetPathReplace** parameter. Please see the usage in the Snippets section in this documentation.

assets/modules/

This is the location of the modules. To change the location:

```
const builder = new ContentBuilder({
    container: '.container',
    modulePath: 'assets/modules/'
});
```

Another location you may want to change is the **plugins/** folder location, which contains plugin-related files.

contentbuilder/plugins/

Here you just need to specify the **pluginPath** parameter with the folder where the **plugins/** folder is located. The default value is the **contentbuilder/** folder. To change the location:

```
const builder = new ContentBuilder({
    container: '.container',
    pluginPath: 'contentbuilder/',
});
```