



#1 Quick start

<https://ckeditor.com/docs/ckeditor5/latest/builds/guides/overview.html>

Table of contents [Classic editor](#) [Example](#) [Inline editor](#) [Example](#) [Balloon editor](#) [Example](#) [Balloon block editor](#) [Example](#) [Document editor](#) [Example](#) [Next steps](#)

Creating an editor using a CKEditor 5 build is very simple and can be described in two steps:

1. Load the desired editor via the `<script>` tag.
2. Call the static `create()` method to create the editor.

There are other installation and integration methods available. For more information check [Installation](#) and [Basic API](#) guides.

#2 Classic editor

In your HTML page add an element that CKEditor should replace:

```
<div id="editor"></div>
```

Load the classic editor build (here [CDN](#) location is used):

```
<script src="https://cdn.ckeditor.com/ckeditor5/23.0.0/classic/ckeditor.js">
</script>
```

Call the `ClassicEditor.create()` method.

```
<script>
  ClassicEditor
    .create( document.querySelector( '#editor' ) )
    .catch( error => {
      console.error( error );
    } );
</script>
```

#3 Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>CKEditor 5 – Classic editor</title>
  <script src="https://cdn.ckeditor.com/ckeditor5/23.0.0/classic/ckeditor.js">
</script>
</head>
<body>
  <h1>Classic editor</h1>
  <div id="editor">
    <p>This is some sample content.</p>
  </div>
  <script>
    ClassicEditor
      .create( document.querySelector( '#editor' ) )
      .catch( error => {
        console.error( error );
      } );
  </script>
</body>
</html>
```

#2 Inline editor

In your HTML page add an element that CKEditor should make editable:

```
<div id="editor"></div>
```

Load the inline editor build (here [CDN](#) location is used):

```
<script src="https://cdn.ckeditor.com/ckeditor5/23.0.0/inline/ckeditor.js">
</script>
```

Call the [InlineEditor.create\(\)](#) method.

```
<script>
  InlineEditor
    .create( document.querySelector( '#editor' ) )
    .catch( error => {
      console.error( error );
    } );
</script>
```

#3 Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>CKEditor 5 - Inline editor</title>
  <script src="https://cdn.ckeditor.com/ckeditor5/23.0.0/inline/ckeditor.js">
</script>
</head>
<body>
  <h1>Inline editor</h1>
  <div id="editor">
    <p>This is some sample content.</p>
  </div>
  <script>
    InlineEditor
      .create( document.querySelector( '#editor' ) )
      .catch( error => {
        console.error( error );
      } );
  </script>
</body>
</html>
```

#2 Balloon editor

In your HTML page add an element that CKEditor should make editable:

```
<div id="editor"></div>
```

Load the balloon editor build (here [CDN](#) location is used):

```
<script src="https://cdn.ckeditor.com/ckeditor5/23.0.0/balloon/ckeditor.js">
</script>
```

Call the [BalloonEditor.create\(\)](#) method.

```
<script>
  BalloonEditor
    .create( document.querySelector( '#editor' ) )
    .catch( error => {
      console.error( error );
    } );
</script>
```

#3 Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>CKEditor 5 – Balloon editor</title>
  <script src="https://cdn.ckeditor.com/ckeditor5/23.0.0/balloon/ckeditor.js">
</script>
</head>
<body>
  <h1>Balloon editor</h1>
  <div id="editor">
    <p>This is some sample content.</p>
  </div>
  <script>
    BalloonEditor
      .create( document.querySelector( '#editor' ) )
      .catch( error => {
        console.error( error );
      } );
  </script>
</body>
</html>
```

#2 Balloon block editor

In your HTML page add an element that CKEditor should make editable:

```
<div id="editor"></div>
```

Load the balloon block editor build (here [CDN](#) location is used):

```
<script src="https://cdn.ckeditor.com/ckeditor5/23.0.0/balloon-
block/ckeditor.js"></script>
```

Call the `BalloonEditor.create()` method.

```
<script>
  BalloonEditor
    .create( document.querySelector( '#editor' ) )
    .catch( error => {
      console.error( error );
    } );
</script>
```

Note: You can configure the block toolbar items using the `config.blockToolbar` option.

#3 Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>CKEditor 5 – Balloon block editor</title>
  <script src="https://cdn.ckeditor.com/ckeditor5/23.0.0/balloon-
block/ckeditor.js"></script>
</head>
<body>
  <h1>Balloon editor</h1>
  <div id="editor">
    <p>This is some sample content.</p>
  </div>
  <script>
    BalloonEditor
      .create( document.querySelector( '#editor' ) )
      .catch( error => {
        console.error( error );
      } );
  </script>
</body>
</html>
```

#2 Document editor

Load the document editor build (here [CDN](#) location is used):

```
<script src="https://cdn.ckeditor.com/ckeditor5/23.0.0/decoupled-
document/ckeditor.js"></script>
```

Call the `DecoupledEditor.create()` method. The decoupled editor requires you to inject the toolbar into the DOM and the best place to do that is somewhere in the promise chain (e.g. one of the `then(() => { ... })` blocks).

The following snippet will run the document editor but to make the most of it check out the [comprehensive tutorial](#) which explains step-by-step how to configure and style the application for the best editing experience.

```

<script>
  DecoupledEditor
    .create( document.querySelector( '#editor' ) )
    .then( editor => {
      const toolbarContainer = document.querySelector( '#toolbar-container'
);
      toolbarContainer.appendChild( editor.ui.view.toolbar.element );
    } )
    .catch( error => {
      console.error( error );
    } );
</script>

```

#3 Example

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>CKEditor 5 – Document editor</title>
  <script src="https://cdn.ckeditor.com/ckeditor5/23.0.0/decoupled-
document/ckeditor.js"></script>
</head>
<body>
  <h1>Document editor</h1>
  <!-- The toolbar will be rendered in this container. -->
  <div id="toolbar-container"></div>
  <!-- This container will become the editable. -->
  <div id="editor">
    <p>This is the initial editor content.</p>
  </div>
  <script>
    DecoupledEditor
      .create( document.querySelector( '#editor' ) )
      .then( editor => {
        const toolbarContainer = document.querySelector( '#toolbar-
container' );
        toolbarContainer.appendChild( editor.ui.view.toolbar.element );
      } )
      .catch( error => {
        console.error( error );
      } );
  </script>
</body>
</html>

```

#2 Next steps

Check the [Configuration guide](#) to learn how to configure the editor — for example, change the default toolbar.

#1 Overview

Table of contents [Available builds](#) [Classic editor](#) [Inline editor](#) [Balloon editor](#) [Balloon block editor](#) [Document editor](#) [Build customization](#) [Additional information](#) [How builds were designed](#) [Use cases](#) [When NOT to use builds?](#)

CKEditor 5 Builds are a set of ready-to-use rich text editors. Every “build” provides a single type of editor with a set of features and a default configuration. They provide convenient solutions that can be installed with no effort and that satisfy the most common editing use cases.

#2 Available builds

The following CKEditor 5 Builds are currently available:

- [Classic editor](#)
- [Inline editor](#)
- [Balloon editor](#)
- [Balloon block editor](#)
- [Document editor](#)

#3 Classic editor

Classic editor is what most users traditionally learnt to associate with a rich text editor — a toolbar with an editing area placed in a specific position on the page, usually as a part of a form that you use to submit some content to the server.

During its initialization the editor hides the used editable element on the page and renders “instead” of it. This is why it is usually used to replace `<textarea>` elements.

In CKEditor 5 the concept of the “boxed” editor was reinvented:

- The toolbar is now always visible when the user scrolls the page down.
- The editor content is now placed inline in the page (without the surrounding `<iframe>` element) — it is now much easier to style it.
- By default the editor now grows automatically with the content.


Heading 1 B I @ :≡ ≡≡ 🖼️ 🗨️ 📄 📺 ↶ ↷

The three greatest things you learn from traveling|

Like all the great things on earth traveling teaches us by example. Here are some of the most precious lessons I've learned over the years of traveling.

Appreciation of diversity

Getting used to an entirely different culture can be challenging. While it's also nice to learn about cultures online or from books, nothing comes close to experiencing cultural diversity in person. You learn to appreciate each and every single one of the differences while you become more culturally fluid.



Leaving your comfort zone might lead you to such beautiful sceneries like this one.

The real voyage of discovery consists not in seeking new landscapes, but having new eyes.

Marcel Proust

To try it out online, check the [classic editor example](#). Jump to [Quick start](#) to start using it.

#3 Inline editor

Inline editor comes with a floating toolbar that becomes visible when the editor is focused (e.g. by clicking it). Unlike classic editor, inline editor does not render *instead* of the given element, it simply makes it editable. As a consequence the styles of the edited content will be exactly the same before and after the editor is created.

A common scenario for using inline editor is offering users the possibility to edit content in its real location on a web page instead of doing it in a separate administration section.

Heading 1 B I @

Gone traveling|

Monthly travel news and inspiration

To try it out online, check the [inline editor example](#). Jump to [Quick start](#) to start using it.

#3 Balloon editor

Balloon editor is very similar to inline editor. The difference between them is that the toolbar appears in a balloon next to the selection (when the selection is not empty):

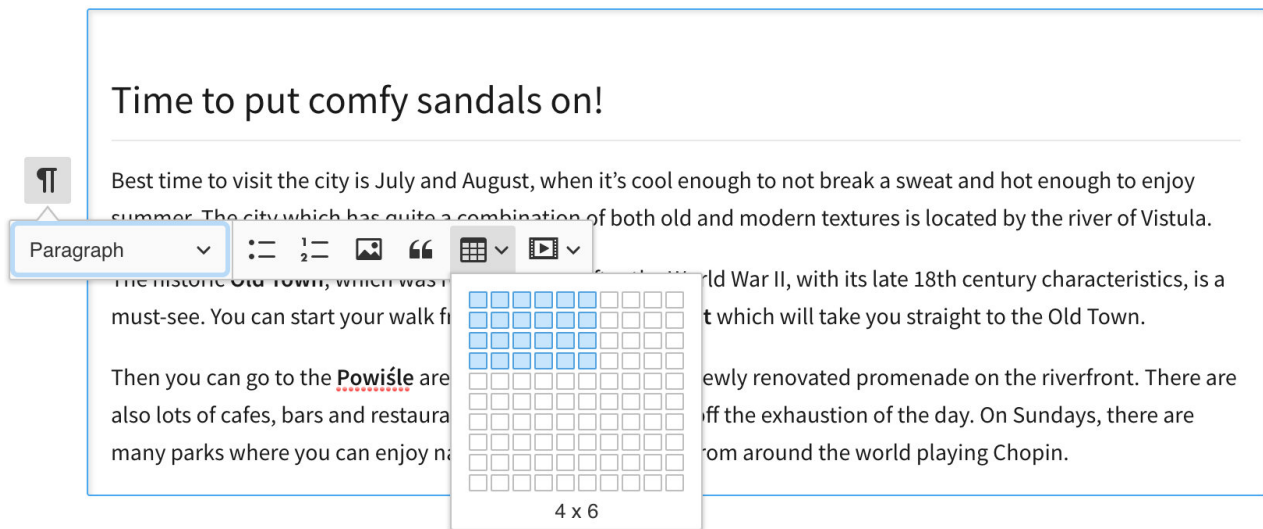


The screenshot shows a document editor interface. At the top, a heading reads "Walking the capitals of Europe: Warsaw". Below it, a paragraph of text is partially visible: "If you enjoyed my previous articles in which we discussed wandering around Copenhagen and Vilnius,". A blue selection box highlights the text "Time to put comfy sandals on!". A white balloon toolbar is positioned above this selection, containing a dropdown menu set to "Heading 2", and icons for bold (B), italic (I), link, bulleted list, numbered list, image, quote, table, video, undo, and redo. To the right of the text is a photograph of a medieval town square with a caption: "Medieval Old Town square, destroyed in 1944 & rebuilt after WWII." Below the photo, another paragraph of text is visible: "Best time to visit the city is July and August, when it's cool enough to not break a sweat and hot enough to enjoy summer. The city which has quite a combination of both old and modern textures is located by the river of Vistula." Further down, another paragraph reads: "The historic Old Town, which was reconstructed after the World War II, with its late 18th century characteristics, is a must-see. You can start your walk from the Nowy Świat Street which will take you straight to the Old Town."

To try it out online, check the [balloon editor example](#). Jump to [Quick start](#) to start using it.

#3 Balloon block editor

Balloon block is essentially the [balloon editor](#) with an extra block toolbar which can be accessed using the button attached to the editable content area and following the selection in the document. The toolbar gives an access to additional, block-level editing features.

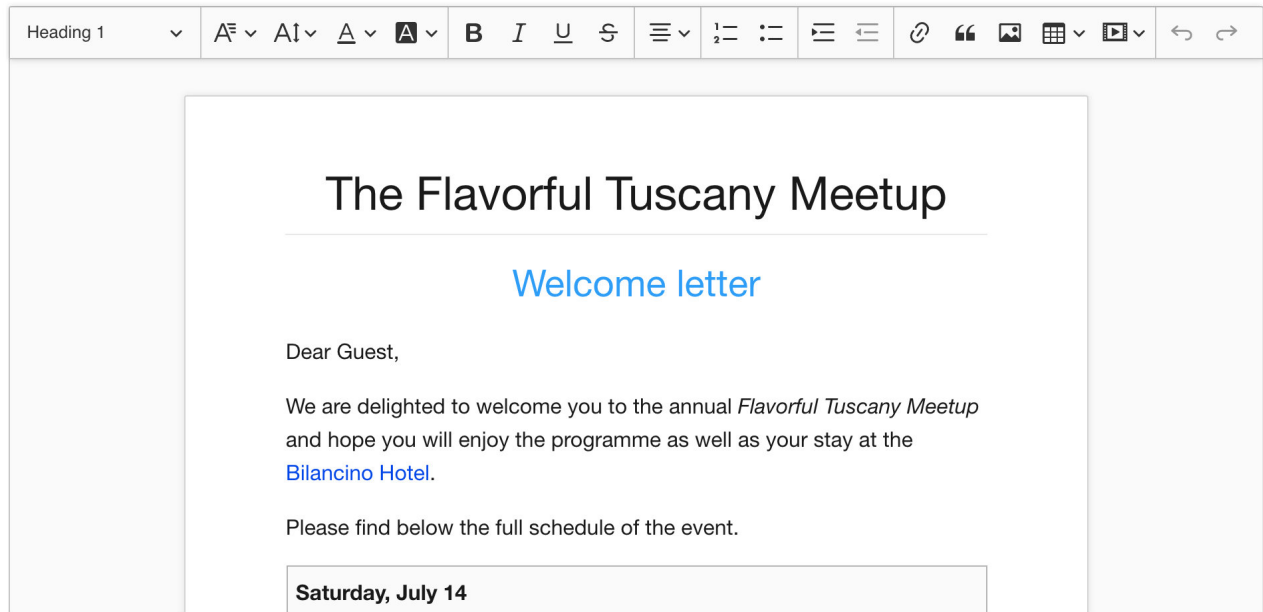


The screenshot shows a document editor interface. A heading reads "Time to put comfy sandals on!". Below it, a paragraph of text is partially visible: "Best time to visit the city is July and August, when it's cool enough to not break a sweat and hot enough to enjoy summer. The city which has quite a combination of both old and modern textures is located by the river of Vistula." A blue selection box highlights the text "Time to put comfy sandals on!". A white balloon toolbar is positioned above this selection, containing a dropdown menu set to "Paragraph", and icons for bulleted list, numbered list, image, quote, table, and video. Below the photo, another paragraph of text is visible: "The historic Old Town, which was reconstructed after the World War II, with its late 18th century characteristics, is a must-see. You can start your walk from the Nowy Świat Street which will take you straight to the Old Town." Further down, another paragraph reads: "Then you can go to the Powiśle area which is a newly renovated promenade on the riverfront. There are many cafes, bars and restaurants here to help off the exhaustion of the day. On Sundays, there are many parks where you can enjoy playing Chopin."

To try it out online, check the [balloon block editor example](#). Jump to [Quick start](#) to start using it.

#3 Document editor

The document editor is focused on rich text editing experience similar to the native word processors. It works best for creating documents which are usually later printed or exported to PDF files.



To try it out online, check the [document editor example](#). Jump to [Quick start](#) to start using it.

#2 Build customization

Every build comes with a default set of features and their default configuration. Although the builds try to fit many use cases, they may still need to be adjusted in some integrations. The following modifications are possible:

- You can override the default **configuration of features** (e.g. define different image styles or heading levels).
- You can change the default **toolbar configuration** (e.g. remove undo/redo buttons).
- You can also **remove features** (plugins).

Read more in the [Configuration guide](#).

If a build does not provide all the necessary features or you want to create a highly optimized build of the editor which will contain only the features that you require, you need to customize the build or create a brand new one. Check [Custom builds](#) for details on how to change the default builds to match your preferences.

#2 Additional information

#3 How builds were designed

Each build was designed to satisfy as many use cases as possible. They differ in their UI, UX and features, and are based on the following approach:

- Include the set of features proposed by the [Editor Recommendations project](#).
- Include features that contribute to creating quality content.
- Provide setups as generic as possible, based on research and community feedback.

#3 Use cases

Each of the builds fits several different use cases. Just think about any possible use for writing rich text in applications.

The following are **some** common use cases:

- In content management systems:
 - Forms for writing articles or website content.
 - Inline writing in a frontend-like editing page.
 - Comments.
- In marketing and sales automation applications:
 - Composing email campaigns.
 - Creating templates.
- In forum applications:
 - Creating topics and their replies.
- In team collaboration applications:
 - Creating shared documents.
- Other uses:
 - User profile editing pages.
 - Book writing applications.
 - Social messaging and content sharing.
 - Creation of ads in recruitment software.

#3 When NOT to use builds?

[CKEditor 5 Framework](#) should be used, instead of builds, in the following cases:

- When you want to create your own text editor and have full control over its every aspect, from UI to features.
- When the solution proposed by the builds does not fit your specific use case.

In the following cases [CKEditor 4](#) should be used instead:

- When compatibility with old browsers is a requirement.
- If CKEditor 4 contains features that are essential for you, which are not available in CKEditor 5 yet.
- If CKEditor 4 is already in use in your application and you are still not ready to replace it with CKEditor 5.

In the following cases [Letters](#) may be used instead:

- When you want an easy way to enable, as part of your application, the creation of articles and documents that feature:
 - Real-time collaborative writing.
 - Inline comments and discussion in the content.
 - Advanced writing features.