MarkdownRenderer-Docs

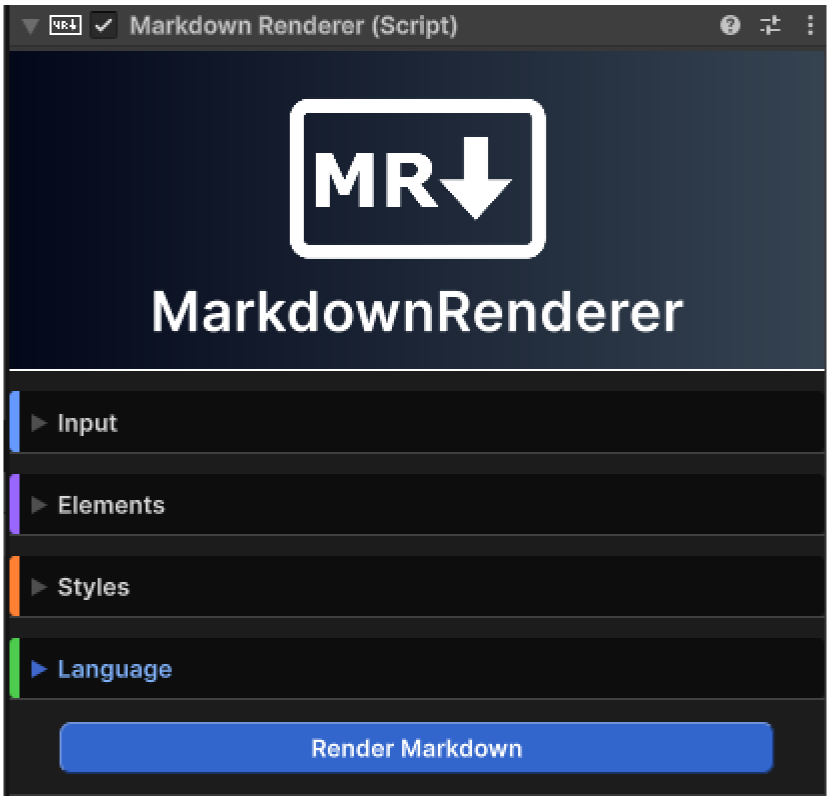
Here's the link to the online documentation:
https://bigcodersplanet.com/docs/markdownrenderer-general/

What is the Markdown Renderer?

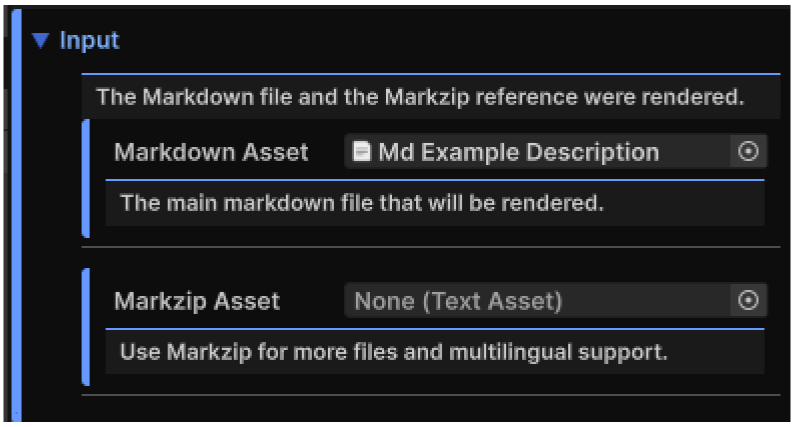
The **Markdown Renderer** is a tool that converts *Markdown*-formatted text into interactive and visually appealing UI elements within a Unity application. **Markdown** is a lightweight markup language that allows you to format and style text using a few simple symbols. The parser processes Markdown input and dynamically generates Unity UI elements, enabling you to embed formatted text, images, links, and more directly into your game's interface.

MarkdownRenderer MonoBehaviour

This is the interface between the Markdown text and the Unity UI. It uses the integrated parser, which is called asynchronously.



Input



The Markdown file and the Markzip reference were rendered.

Markdown Asset: Md Example Description

The main markdown file that will be rendered.

Markzip Asset: None (Text Asset)

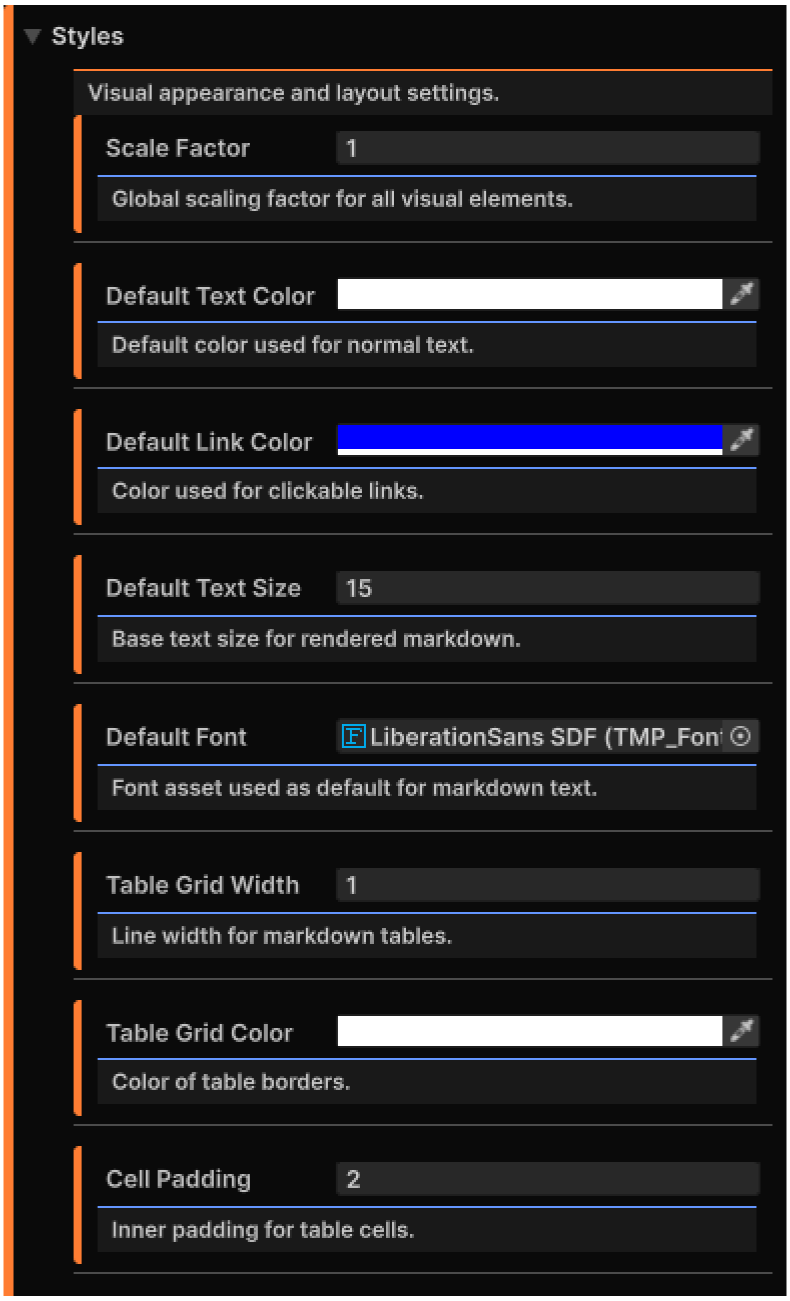
Use Markzip for more files and multilingual support.

Here you define which Markdown content will be rendered.
The **Markdown Asset** is a simple text asset and can be directly assigned from a `.md` file.
The **Markzip Asset** is created by a preprocessing step to include external content. More about this in the *Markzip* chapter.

You can also call the asset and render function directly via C#:

```
using MarkdownToUnity;
public MarkdownRenderer markdownRenderer;
markdownRenderer.Render(TextAsset obj);
```

Styles



Styles

Visual appearance and layout settings.

Scale Factor: 1
Global scaling factor for all visual elements.

Default Text Color
Default color used for normal text.

Default Link Color
Color used for clickable links.

Default Text Size: 15
Base text size for rendered markdown.

Default Font: LiberationSans SDF (TMP_Fon
Font asset used as default for markdown text.

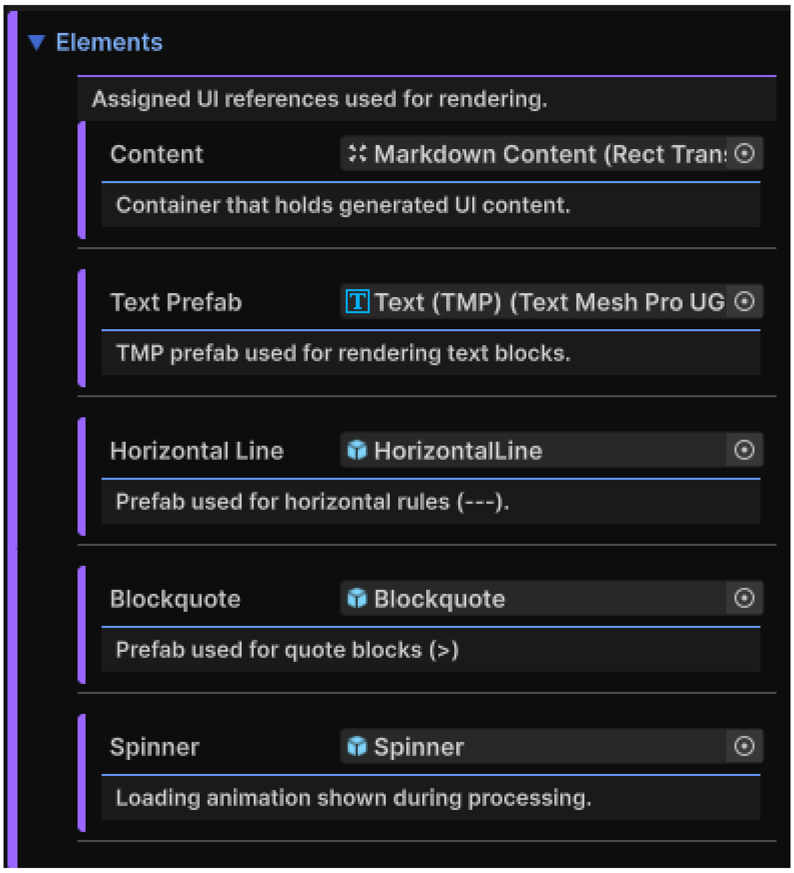
Table Grid Width: 1
Line width for markdown tables.

Table Grid Color
Color of table borders.

Cell Padding: 2
Inner padding for table cells.

Setting	Description
Visual Element Scale Factor	Adjusts the size of visual elements (e.g., images and videos) after rendering. Depending on the desired overall size, elements can be scaled larger or smaller.
Default Text Color	Sets the default color of the text.
Default Text Size	Defines the default text size.
Default Font	Specifies the default font.
Table Grid Width	Sets the line width of table borders.
Table Grid Color	Sets the line color of table borders.
Cell Padding	Defines the inner padding of table cell contents.

Elements


Elements

Assigned UI references used for rendering.

Content: Markdown Content (Rect Tran:

Container that holds generated UI content.

Text Prefab: Text (TMP) (Text Mesh Pro UG

TMP prefab used for rendering text blocks.

Horizontal Line: HorizontalLine

Prefab used for horizontal rules (---).

Blockquote: Blockquote

Prefab used for quote blocks (>)

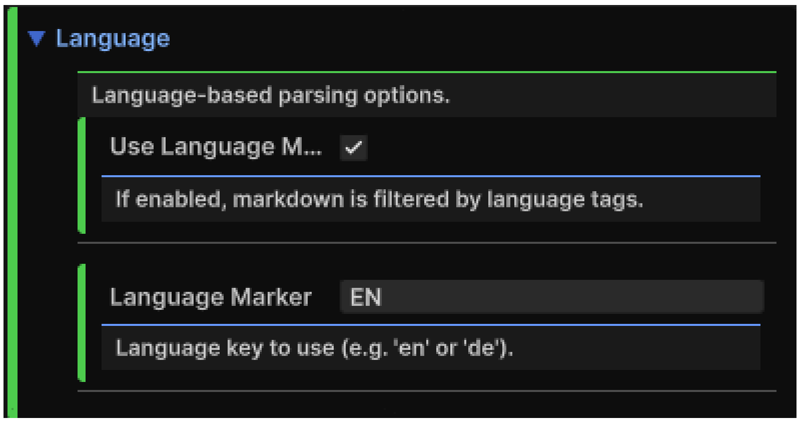
Spinner: Spinner

Loading animation shown during processing.

These elements are instantiated during rendering.
Adjust objects such as horizontal lines, text, or blockquotes to your needs — they serve as templates used to generate the final UI elements.

Replace the spinner to display the loading state differently. It is only visible for larger compressed Markzip files.

Multilanguage Support


Language

Language-based parsing options.

Use Language M...: ✓

If enabled, markdown is filtered by language tags.

Language Marker: EN

Language key to use (e.g. 'en' or 'de').

Set **Use Language Marker** to *true* if you want to use multiple `.md` versions in different languages.
Save the translations in the same folder as all other resources. Since the paths to external files are identical to those of the translated files, they will be used automatically.

Add a unique language marker to the filename so it can be detected. For example: `EN`, `GER`, `ESP`, etc.
If no matching file is found, the first `.md` file will be displayed, or a file with the default marker, which is set to `"EN"` by default.
You can change this default marker in the `MarkdownToUnityParser` class.

The current language is represented by the **Language Marker** of the MarkdownRenderer.

Integrating Markdown

Three variants are supported for integrating Markdown files, each with different features.

MarkdownAsset

A **MarkdownAsset** refers to the simple integration of a native `.md` file into the MarkdownRenderer.

How to use it

Simply drag the Markdown file into Unity and assign it directly to the MarkdownRenderer as a reference.
Alternatively, call the `Render()` function from an external script and pass the `.md` file directly as a parameter, as Unity interprets `.md` files as `TextAsset`.

Features:

- Headers (#)
- Bold (**text**), *Italics* (*text*)
- Lists (- item)
- Blockquotes (> text)
- Internal Images (base64)
- Internet Images via URL
- Tables
- Links

Advantages:

- Direct integration of `.md` files without conversion
- Native parsing of text content without transformations (faster)

Disadvantages:

- Does not support external files such as images or videos
- No multilingual support

Use case: Simple text with internal images in base64 format

MarkdownZip

A **MarkdownZip** contains a folder where external resources like videos, images, and the actual `.md` files are stored in a compressed format.

Features:

- Headers
- Bold (**text**), *Italics* (*text*)
- Lists (- item)
- Blockquotes (> text)
- Internal and external files
- Internet Images via URL
- Tables
- Links

Advantages:

- External files do not need to be interpreted as base64 strings and are stored temporarily
- Supports multilingual content
- Resources are compressed in a single file instead of being separate Unity assets

Disadvantages:

- Slower, but works asynchronously
- Extra compression step required

How to use it

Create the Markdown file and include external resources. These must reside in a subfolder together with the `.md` file.
This folder is then converted in Unity using our conversion tool. The resulting `.bytes` file can be used as a `TextAsset` with the MarkdownRenderer.

Use case: Text with both internal and external files and multilingual support

---

Markbook

Markbook allows bundling multiple MarkdownZip files, enabling them to be accessed as individual chapters.

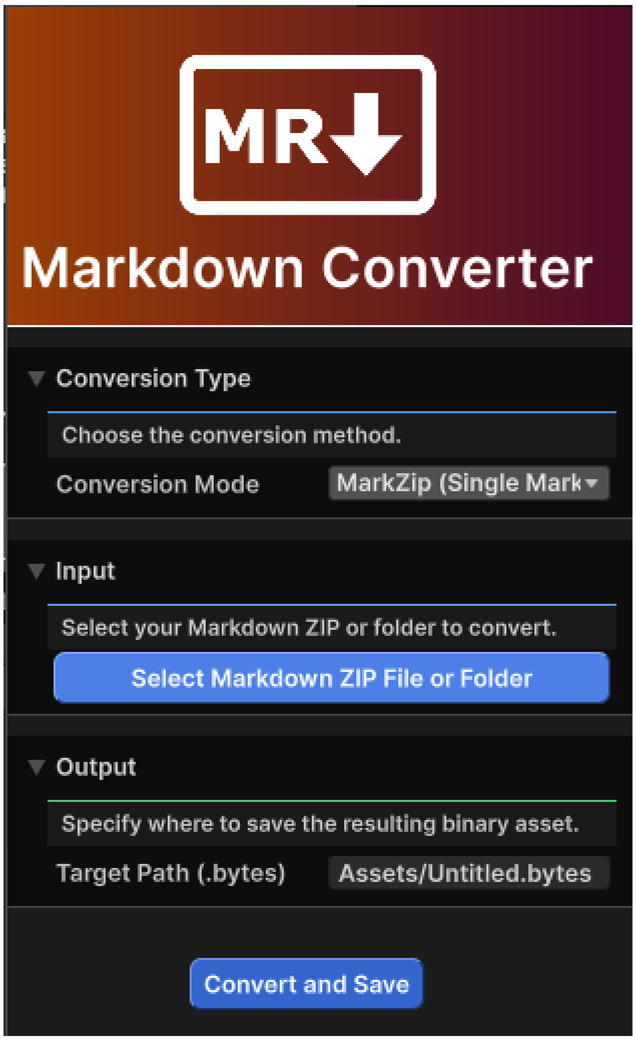
This is useful for combining different documents easily.

Use case: Multiple chapters with internal and external files, supporting multilingual content

---

Markdown Converter Editor

The converter allows you to create MarkdownZips and Markbooks.
Go to **Window -> MarkdownToUnityUI -> Convert Markdown to Asset**.



Choose the conversion method.

Conversion Mode: MarkZip (Single Mark

Select your Markdown ZIP or folder to convert.

Specify where to save the resulting binary asset.

Target Path (.bytes): Assets/Untitled.bytes

MarkdownZip

1. Select **Markzip** in the dropdown.
2. Click the button **"Select Markdown ZIP File or Folder"**. You will be prompted to choose either an existing zip archive or an uncompressed folder containing the `.md` file and external resources.
3. Select the target path. The easiest way is to right-click a folder in Unity, choose **Copy Path**, and paste it. A local path within the Unity project is required. Name the file.

Markbook

1. Select **Markbook** in the dropdown.

2. Click the button **"Select Markdown ZIP File or Folder"**. Choose the parent folder containing all the Markdown chapters. Each subfolder contains the `.md` file and its external resources for the chapter.
3. Select the target path. The easiest way is to right-click a folder in Unity, choose **Copy Path**, and paste it. A local path within the Unity project is required. Name the file.

The resulting binary `TextAsset` can then be used with the MarkdownRenderer.
A Markbook must be loaded first. Afterwards, the subchapters can be accessed directly in the MarkdownRenderer or by their chapter name.

Parser

You can also manually use the parser to convert a Markdown text into a TextAsset or a string in RichText format to display it with TextMeshPro.

```
using MarkdownToUnity;
UnityUIMarkdownTextAsset asset =
MarkdownUnityParser.ParseMarkdown(markdownText);
```

The asset contains all the elements to be rendered. Example functions can be found in the `MarkdownToUnityRenderer` class.

For asynchronous calls, you can use coroutines or **UniTask**:

```
private IEnumerator Test(TextAsset markZip, string languageMarker) {

            if(spinner != null) { spinner.SetActive(true); }
            Task<UnityUIMarkdownTextAsset> parseTask =
MarkdownUnityParser.ParseMarkdown(markZip, languageMarker);

            yield return new WaitUntil(() => parseTask.IsCompleted);

            if (parseTask.IsFaulted) {
                Debug.LogError($"Exception:
{parseTask.Exception?.InnerException?.Message}");
            } else {
                UnityUIMarkdownTextAsset asset = parseTask.Result;
                if (asset != null) {
                    MarkdownUnityRenderer.Render(asset, content,
textPrefab.gameObject, horizontalLinePrefab, blockquotePrefab);
// Or own Logic
                }

            }
            if (spinner != null) { spinner.SetActive(false); }
```

```
        }
        }
```