

LeBlender

Documentation



v 1.0.x

[Presentation](#)

[How to install](#)

[Upgrading to 1.0.0](#)

[LeBlender editors manager](#)

[Presentation](#)

[Extend LeBlender editor manager](#)

[LeBlender Editor](#)

[Presentation](#)

[LeBlenderModel](#)

[LeBlenderModel Properties](#)

[LeBlenderValue Methods](#)

[LeBlender Editor Caching](#)

[Custom Controller](#)

Presentation

LeBlender is an open source Umbraco backoffice extension which made the Grid Canvas Editors management easier and flexible.

We can create, order, update, remove and extend Grid editors on the fly, through a very simple and nice user interface.

LeBlender project brings two main features:

- **LeBlender Editors Manager**: UI for Grid editors management.
- **LeBlender Editor**: Super powerful editor for advanced editor creation on the fly.

How to install

1. Download LeBender Package
<https://our.umbraco.org/projects/backoffice-extensions/leblender>
2. Install it through the Umbraco Backend

Upgrading to 1.0.0

We have tried to make LeBlender 1.0.0 compatible with previous versions.

Nevertheless there are some **breaking changes** regarding the LeBlender editor. The most relevant is that the default LeBlender model name has changed. On the previous version it was called **BlenderModel**, it has been changed for **LeBlenderModel**.

Follow those steps to upgrade your project

1. Remove LeBlender Datatype
2. Uninstall previous version (delete "/App_plugin/Lecoati.LeBlender/" folder and "/bin/Lecoati.leblender.Extension.dll")
3. Install LeBlender 1.0.0
4. Change your BlenderModel references to LeBlenderModel
5. Save and publish the content which uses the LeBlender editor

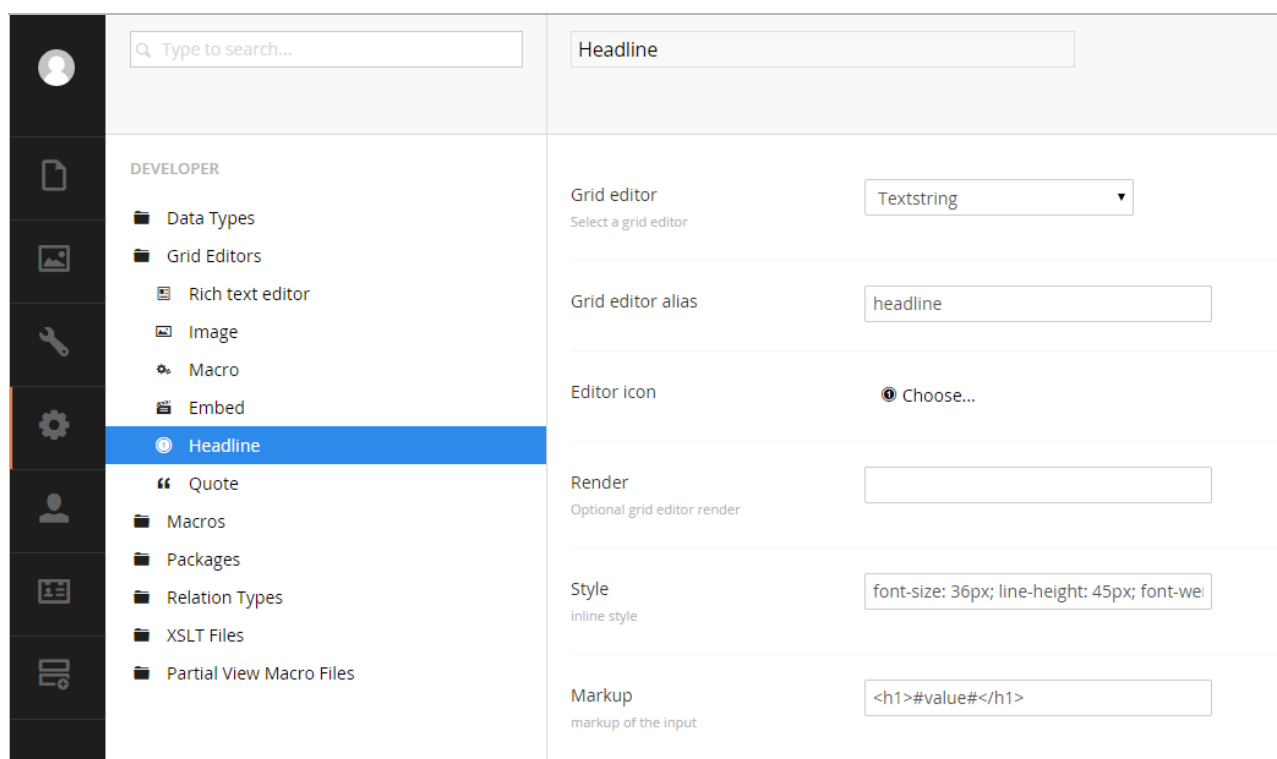
LeBlender editors manager

Presentation

This part of the project is a user interface tool for managing the grid editor settings.

Basically, it's a simple UI to edit the grid config file `grid.editors.config.js`. Through it, we can manage our Umbraco project's set of grid editors without manually writing any line of JSON code.

In addition, in this new version, we made it possible to extend it with your own grid editor in a very easy way, see next section.



Type to search...		Headline
DEVELOPER		
Data Types		
Grid Editors		
Rich text editor		
Image		
Macro		
Embed		
Headline		
Quote		
Macros		
Packages		
Relation Types		
XSLT Files		
Partial View Macro Files		
Grid editor	Select a grid editor	Textstring
Grid editor alias		headline
Editor icon		Choose...
Render	Optional grid editor render	
Style	inline style	font-size: 36px; line-height: 45px; font-weight: bold
Markup	markup of the input	<h1>#value#</h1>

Extend LeBlender editor manager

By default, Umbraco Grid Canvas comes with 5 editors: rte, image, macro, embed and textstring.

We did possible to extend this list with your own editors or others from the community in a very simple way.

It can be done the same way as we declare a property editor for Datatype into a **package.manifest**, we just have to specify that the property editor is a Grid Editor **"isGridEditor": true**.

In addition, we can use prevalues to define the needed fields for our editor configuration.

Let's see for instance the textstring editor configuration:

```
{
  "propertyEditors": [
    {
      "name": "Textstring",
      "alias": "textstring",
      "isGridEditor": true,
      "editor": { "view": "textstring" },
      "prevalues": {
        "fields": [
          {
            "label": "Style",
            "key": "style",
            "description": "inline style",
            "view": "textstring",
          },
          {
            "label": "Markup",
            "key": "markup",
            "description": "markup of the input",
            "view": "textstring",
          }
        ]
      }
    }
  ]
}
```

In this case, two prevalues are needed for this editor: **Styles** and **Markup**

The screenshot shows the Umbraco configuration interface for a 'Headline' editor. The left sidebar lists various editor types, with 'Headline' selected. The main configuration area includes the following fields:

- Grid editor alias:** headline
- Editor icon:** Choose...
- Render:** Optional grid editor render
- Style:** font-size: 36px; line-height: 45px; font-weight: b (inline style)
- Markup:** <h1>#value#</h1> (markup of the input)

The 'Style' and 'Markup' fields are highlighted with a green box, indicating they are the two prevalues needed for this editor. A green 'Save' button is located at the bottom of the configuration area.

All prevalue editors stored in **/umbraco/views/prevalueeditors/** can be used as prevalue field or you can use your custom editor as well.

More information about property editor: <https://our.umbraco.org/...rs/property-editors-v7>

LeBlender Editor

Presentation

When we start a new project with the Umbraco Grid Layout, 5 different editors are available by default: RTE, image, macro, embed and textstring.

We have added a new one: **LeBlender Editor**

This new grid editor allow us to create complex data structures for our Grid in just a few clicks without any line of code.

It is a perfect solution for sliders, carousels, tabs, highlighted content and so much more...

Its main features are:

- 100% configurable by UI
- Optional preview within the grid backend property editor
- Simple set of properties or list of them
- Any datatype can be used as LeBlenderEditor properties
- LeBlenderEditor can be cached
- Custom controllers can be used

The screenshot shows the Umbraco Grid Layout backend interface. On the left is a sidebar with a search bar and a list of developer tools. The 'Carousel' editor is selected in the main panel. The configuration fields include:

- Grid editor:** A dropdown menu set to 'LeBlender Editor'.
- Grid editor alias:** A text field containing 'carousel'.
- Editor icon:** A button labeled 'Choose...'.
- Render:** A text field for the 'Optional grid editor render'.
- Properties:** A section titled 'LeBlender Editor properties' with a list of properties: 'Image', 'Title', and 'Summary'. There is an '+ Add property' button.
- Render in the grid:** A checkbox that is checked.
- Min:** A text field containing '1', with the label 'Minimum number of items' below it.

On the right side, there is a 'Property setting' panel for the 'Image' data type, showing fields for 'Name' (set to 'Image'), 'Alias' (set to 'image'), and 'Description'.

On the frontend side, by default the **LeBlender** engine looks in the folder **/views/partials/grid/editors** for a partial view with the same name as the editor. You can of course define your own custom path instead, through the **Render** field.

LeBlenderModel

The partial view will receive a typed object of type **LeBlenderModel** that you can use to easily access the editor's data:

```
@inherits UmbracoViewPage<Lecoati.LeBlender.Extension.Models.LeBlenderModel>

@foreach (var item in Model.Items)
{
    <div>
        
        <h3>@item.GetValue("title")</h3>
        <p>@item.GetValue("summary")</p>
        <p><a href="@Umbraco.TypedContent(item.GetValue<string>("link")).Url">Learn more</a>
    </div> }
```

LeBlenderModel Properties

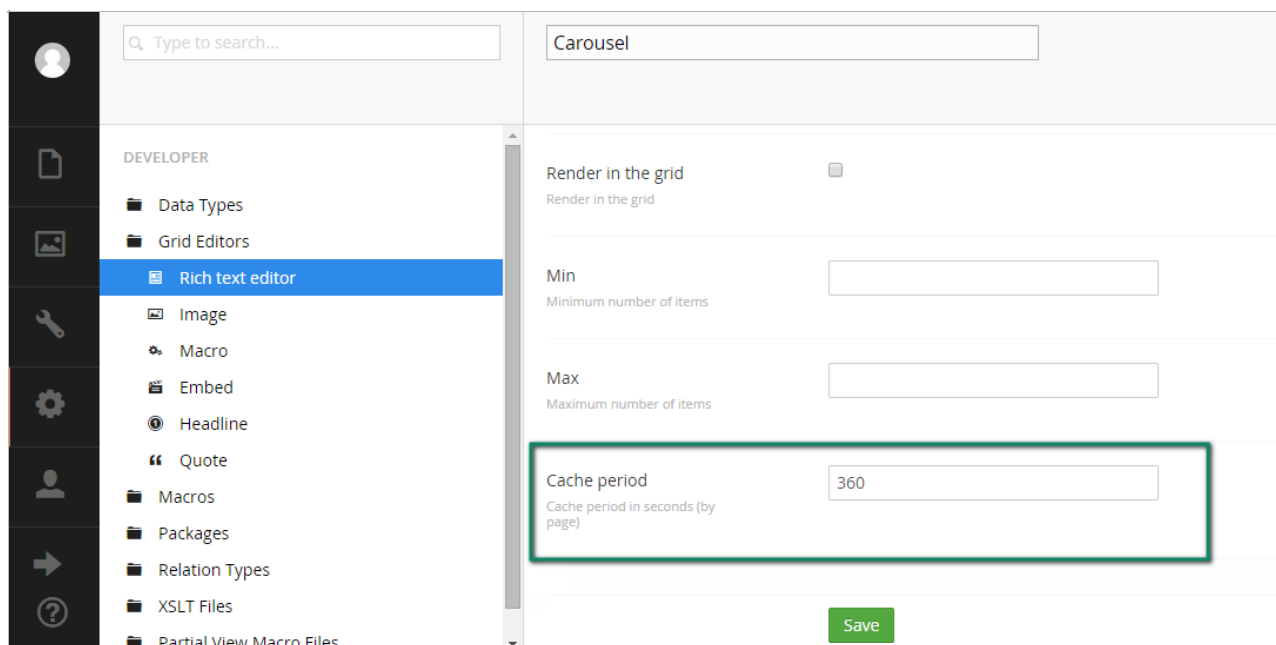
Name	Description
Items	IEnumerable of LeBlenderValue

LeBlenderValue Methods

Name	Parameters	Description
GetValue	string propertyAlias	Get the property value as a string by Alias
GetValue<T>	string propertyAlias	Get the property value as T by Alias

LeBlender Editor Caching

LeBlender Editors can be **cached**. Cache duration in second and by page can be specified within the advanced LeBlender Editor's settings:



The LeBlender Editor cache is refreshed every time the content is published, so your changes are rendered immediately.

Custom Controller

Because our grid editors sometimes need some logic and it's always better to isolate it within controllers, custom controllers can be created for the LeBlender Editors.

To do it, we just need to create a standard MVC controller that inherits from **LeBlenderController** with the same name as our LeBlender Editor.

In the following example, the editor is called **LastTweets**:

```
public class LastTweetsController : LeBlenderController
{
    public ActionResult Index(LeBlenderModel model)
    {
        // Do your stuff here
        // ...
        return View(model);
    }
}
```

In this case, we are routing the action to the default **Index** method, but we have also allowed routing to different actions based on the partial view name that is being rendered.

In the following example, the partial view is called **DisplayLastTweets**:

```
public class LastTweetsController : LeBlenderController
{
    public ActionResult DisplayLastTweets(LeBlenderModel model)
    {
        // Do your stuff here
        // ...
        return View(model);
    }
}
```

The controller receives a **LeBlenderModel** object and can return any custom model to the partial view.