Retrieve structured content in rich text

January 24, 2023 • Martina Farkasova • 10 min read • .NET

There are times when you want to make sure your content is future-proof and can be used outside individual pages. In such cases, the structure of your content matters, as you can read about in our blog post about WYSIWYG page builders.

In this tutorial, you'll learn how to resolve content components and content items inserted in your rich text elements. Also, you'll see how to resolve hyperlinks to content items.

To understand how to deal with structured data models, you'll also learn how to add blockquotes to articles.

How structured rich text helps you

One of the benefits of using an API-first CMS like Kontent.ai is that you're able to clearly structure your content so it can be used in any channel. But you don't want to be restricted to needing to know exactly what content will be included ahead of time.

Structure is great, but your content editors want the freedom to add various amounts of various kinds of content. For this, you can add structure to your rich text with content components and linked items.

The major difference between content components and linked items is that components exist only inside a single content item, while linked items can be reused across your project. You can read more about when to use components and when linked items.

With Kontent.ai, you can link content items together in two ways:

1. Using a linked items element – here, you can limit the items, including what types can be included and how many. Read how to retrieve linked content items.
2. In a rich text element – here, your items are included within the text, which requires additional work from your app to render them. This way is covered here.

This article will look at how to add structure into your rich text through examples of components. For your rich text, the principle is the same for components and linked items. The items and components can be distinguished in the JSON response, but it's not necessary for these purposes.

There are two ways to implement the structure on your front end:

1. Defining resolvers for content types that might appear in the rich text.
2. Using templates or specific views for structured blocks.

With most technologies, you can choose whether to take a global approach with resolvers or iterate over blocks, such as by making use of design templates for blocks within your app.

Choosing to use a structured model can help ensure you don't need to know what type of content is being added to your rich text as your hierarchy will be created automatically.

Resolve items and components in rich text

This example shows how to resolve tweets embedded in articles. The tweets can be inserted either as components or content items. Once you have tweets in your content, follow the steps below to resolve the tweets in your code. You can use this approach for any other type of structured content in your rich text elements.

Without adjusting your application, any content item or component inserted in a rich text element will resolve to an empty object reference, which won't be rendered on the page.

HTML

```
<object type="application/kenticocloud" data-type="item" data-codename="my_tweet">
</object>
```

Tip: If you're using strongly typed models, remember to generate a model for your *Article* and *Tweet* content types.

1. Implement a tweet resolver

To ensure your tweet is resolved exactly as you'd like it, create a Tweet resolver for rich text elements.

C#

```
public class TweetResolver : IInlineContentItemsResolver<Tweet>
    {
        public string Resolve(Tweet data)
        {
            return
                $"<blockquote class=\"twitter-tweet\" data-lang=\"en\" data-theme=\"
{data.Theme}\"><a href=\"{data.TweetLink}\"></a></blockquote>";
        }
    }
```

2. Register the resolver

You'll need to register the resolver to ensure it's used.

C#

```
// You can also register it in IServiceCollection or another framework for dependency
injection: https://github.com/kontent-ai/delivery-sdk-
net/blob/master/docs/customization-and-extensibility/rich-text/string-based-linked-
items-rendering.md#registering-a-resolver

using Kontent.Ai.Delivery;
```

```
using Kontent.Ai.Delivery.InlineContentItems;

IDeliveryClient client = DeliveryClientBuilder
    .WithProjectId("<YOUR_PROJECT_ID>")
    // Registers a content item resolver for tweets
    .WithInlineContentItemsResolver(new TweetResolver())
    // Registers the generated strongly typed models
    .WithTypeProvider(new CustomTypeProvider())
    .Build();
```

3. Retrieve the rich text

Now that your app knows how to resolve strings that contain tweets in them, it's enough to retrieve an article and the tweet will be included in the body.

C#

```
IDeliveryItemResponse response = await client.GetItemAsync<SimpleArticle>("my_article");
SimpleArticle simpleArticle = response.Item;

string simpleArticleBody = simpleArticle.Body;
```

For the final HTML, you could also work with embedded tweets, such as by calling the Twitter API.

Resolve hyperlinks to content items

This example will show you how to resolve links to content item used within the body of articles. Without adjusting your application, any link in a rich text element that points to a content item will contain an empty value. Let's see how to resolve such links correctly.

1. Define a model

If you're using strongly typed models, add a model for each type of content item you want to link within rich text.

2. Implement a resolver

To ensure your links are resolved correctly, you need to implement a resolver with two methods:

- The first method will return the URL of an available item.
- The second method will return a 404 Not found error when the linked item is not available.

A content item is unavailable when deleted or, in case of live environments, unpublished.

C#

```
public class CustomContentLinkUrlResolver : IContentLinkUrlResolver
{
    public Task<string> ResolveLinkUrlAsync(IContentLink link)
    {
```

```
        // Resolves URLs to content items based on the Article content type
        if (link.ContentTypeCodename == "article")
        {
            return Task.FromResult($"/articles/{link.UrlSlug}");
        }

        // TODO: Add the rest of the resolver logic
    }

    public Task<string> ResolveBrokenLinkUrlAsync()
    {
        // Resolves URLs to unavailable content items
        return Task.FromResult("/404");
    }
}
```

When building the resolver logic, you can use the `link` parameter to get more information about the linked content items.

3. Register the resolver

You'll need to register the resolver to ensure its use.

C#

```
// You can also register the resolver in IServiceCollection or another framework for dependency injection: https://kontent.ai/learn/net-register-resolver
using Kontent.Ai.Delivery;

IDeliveryClient client = DeliveryClientBuilder
    .WithProjectId("<YOUR_PROJECT_ID>")
    // Registers the resolver
    .WithContentLinkUrlResolver(new CustomContentLinkUrlResolver())
    .Build();
```

4. Retrieve the article

Now that your app knows how to resolve links, it's enough to simply retrieve an article and the links will be included inside the body.

C#

```
IDeliveryItemResponse response = await client.GetItemAsync<SimpleArticle>("my_article");
SimpleArticle simpleArticle = response.Item;

string simpleArticleBody = simpleArticle.Body;
```

The URL to a content item linked in the rich text element is now correctly resolved.

HTML

```
<p>The used coffee grounds are retained in the filter, while the <a
href="/articles/which-brewing-fits-you" data-item-id="65832c4e-8e9c-445f-a001-
b9528d13dac8">brewed coffee</a> is collected in a vessel such as a carafe or pot.</p>
```

Adding blockquotes

This example will go through how to add blockquotes to articles within MVC apps.

You can read about how to use a structured data model for .NET MVC projects to give you an idea how you can work with display templates (a similar approach could be used with an engine such as Thymeleaf with the Java Delivery SDK, as you can see in a video about building web services with the Java Delivery SDK and Spring Boot on our blog).

Modeling and creating blockquotes in Kontent.ai

First, it's necessary to model your content in Kontent.ai. You'll need a Blockquote content type and then to add a blockquote to a Simple Article. The process is similar to the example of inserting tweets into blog posts.

For the Blockquote content type, it's enough if it has two fields – one for the quoted text and one for quote's source. You can use a Rich text element for the quote itself in case you want to include links or styling and a Text element for the source. When you're done, it might look like the image below.

How your Blockquote type might look

You can then add a blockquote to a Simple Article as a component. It might then look like the image below.

How an article might look with a blockquote in it as a component

If you made a simple API call to retrieve this article, the response would look something like this (shortened for clarity):

JSON

```
{
  "item": {
    "system": {
      "id": "f9b0fd1c-1b83-491a-9d64-2737faedf80d",
      "name": "Bourbon Coffee",
      "codename": "bourbon_coffee",
      "language": "en-US",
      "type": "simple_article",
      "sitemap_locations": [],
      "last_modified": "2019-01-11T11:46:18.2473895Z"
    },
    "elements": {
      "title": {
        "type": "text",
        "name": "Title",
        "value": "Bourbon Coffee"
      },
      "author": {
        ...
      },
      "body": {
        "type": "rich_text",
        "name": "Body",
        "images": {},
        "links": {},
        "modular_content": [
          "n44bfddb7_b088_01ef_e782_423deb064718"
        ],
```

```
        "value": "<p>Arabica Bourbon is among the best coffee varieties you can find in Brazil, Salvador, and Rwanda. This widely known and popular coffee is cultivated in three color varieties: red, orange, and yellow. But what does it have in common with the American whiskey? </p>\n<p>The coffee, first called Café du Roy, then Café Leroy, was served to kings at the French court and was the brand of choice of the classic author, Honoré de Balzac, who enjoyed forty cups a day. Or so they say…</p>\n<object type=\"application/kenticocloud\" data-type=\"item\" data-rel=\"component\" data-codename=\"n44bfddb7_b088_01ef_e782_423deb064718\"></object>\n<p><br></p>"
      }
    }
  },
  "modular_content": {
    ...
    "n44bfddb7_b088_01ef_e782_423deb064718": {
      "system": {
        "id": "44bfddb7-b088-01ef-e782-423deb064718",
        "name": "44bfddb7-b088-01ef-e782-423deb064718",
        "codename": "n44bfddb7_b088_01ef_e782_423deb064718",
        "language": "en-US",
        "type": "blockquote",
        "sitemap_locations": [],
        "last_modified": "2019-01-11T11:46:18.2473895Z"
      },
      "elements": {
        "quote": {
          "type": "rich_text",
          "name": "Quote",
          "images": {},
          "links": {},
          "modular_content": [],
          "value": "<p>If I couldn't, three times a day,</p>\n<p>be allowed to drink my little cup of coffee,</p>\n<p>in my anguish I will turn into</p>\n<p>a shriveled-up roast goat. </p>"
        },
        "source": {
          "type": "text",
          "name": "Source",
          "value": "Coffee Cantata by J. S. Bach"
        }
      }
    }
  }
}
```


You have both your main text and your blockquote to add. Now it's a matter of telling your app how to handle the structure you've added.

Modeling blockquotes in your app

If you're using strongly typed models, you should remember to add your Blockquote model.

C#

```
// Generate strongly typed models at https://github.com/kontent-ai/model-generator-net

using System;
using System.Collections.Generic;
using Kontent.Ai.Delivery.Abstractions;

namespace KontentAiModels
{
    public partial class Blockquote
    {
        public const string Codename = "blockquote";
        public const string QuoteCodename = "quote";
        public const string SourceCodename = "source";

        public IRichTextContent Quote { get; set; }
        public string Source { get; set; }
        public IContentItemSystemAttributes System { get; set; }
    }
}
```

You can see example Simple Article models in Retrieving linked content. For .NET models, you need to make one change – you need to type the Body property as `IRichTextContent` (not `string`).

Controlling how blockquotes will look

This will depend on the technology you are using. See some examples below. The actual appearance of the blockquote can be defined separately, such as through CSS for a website.

HTML

```
<!-- Place this in a file at a path similar to:
Views/Articles/DisplayTemplates/Blockquote.cshtml -->

@model DancingGoat.Models.Blockquote

@{
    <blockquote>@(Model.Quote) - <cite>@(Model.Source)</cite></blockquote>
}
```

What's next?

Check out how to render different output for rich text in ASP.NET MVC, a blog post by Rob West. Also, see our docs on how to approach rendering structured rich text and strings in rich text in .NET apps.

- Link content together outside your rich text.
- Import rich text via the Management API.
- Read more about linked items and components in our API reference.