Retrieve structured content in rich text

January 24, 2023 • Martina Farkasova • 10 min read • JavaScript

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.

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.

To ensure your tweet component is resolved exactly as you'd like it, you need to define the HTML markup for the tweet in your rich text resolver.

For resolving content components and content items inserted in the rich text element, the SDK provides the `resolveRichText()` function with the `contentItemResolver` parameter. Within the `contentItemResolver`, you need to specify how to resolve components or items of a given type. In this example, you specify how to resolve tweets.

JavaScript

```
const KontentDelivery = require('@kontent-ai/delivery-sdk');

// Initializes the Delivery client
const deliveryClient = KontentDelivery.createDeliveryClient({
  environmentId: '<YOUR_ENVIRONMENT_ID>',
});

// Gets your content item
const response = await deliveryClient.item('my_article')
  .toPromise();

// Stores the contents of the rich text element
const richTextElement = response.data.item.elements.body;

// Note: The code below executes correctly in browser. To adjust the code for nodejs,
see https://kontent.ai/learn/js-rte-nodejs
// Defines how to resolve the rich text element
const resolvedRichText = KontentDelivery.createRichTextHtmlResolver().resolveRichText({
  // Gives the resolver the contents of your rich text
  element: richTextElement,
  // Points the resolver to the content items and components that might be used in the
```

```
rich text element
  linkedItems:
KontentDelivery.linkedItemsHelper.convertLinkedItemsToArray(response.data.linkedItems),
  contentItemResolver: (contentItem) => {
    // For tweet items and components, resolves to the following HTML markup
    if (contentItem && contentItem.system.type === 'tweet') {
      return {
        contentItemHtml: `<blockquote class="twitter-tweet" data-lang="en" data-
theme="${contentItem.elements.theme.value}"><a
href="${contentItem.elements.tweetLink.value}"></a></blockquote>`
      };
    }

    // For other type of items and components, resolves to an empty string
    return {
      contentItemHtml: `<div>Unsupported type ${contentItem.system.type}</div>`
    };
  }
});

// Gets the resolved HTML content of your rich text
const resolvedRichTextHtml = resolvedRichText.html;
```

You can also see the resolvers in our sample React and Vue applications.

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.

To resolve hyperlinks that point to content items, you need to implement a rich text URL resolver in your code.

For resolving hyperlinks to content items in the rich text element, the SDK provides the `resolveRichText()` function with the `urlResolver` parameter. Within the `urlResolver`, you need to specify how to resolve links to items of a given type. In this example, you specify how to resolve links to articles.

JavaScript

```
const KontentDelivery = require('@kontent-ai/delivery-sdk');

// Initializes the Delivery client
const deliveryClient = KontentDelivery.createDeliveryClient({
  environmentId: '<YOUR_ENVIRONMENT_ID>',
});
```

```

// Gets your content item
const response = await deliveryClient.item('my_article')
  .toPromise();

// Stores the contents of the rich text element
const richTextElement = response.data.item.elements.body;

// Note: The code below executes correctly in browser. To adjust the code for nodejs, see https://kontent.ai/learn/js-rte-nodejs
// Defines how to resolve the rich text element
const resolvedRichText = KontentDelivery.createRichTextHtmlResolver().resolveRichText({
  // Gives the resolver the contents of your rich text
  element: richTextElement,
  urlResolver: (linkId, linkText, link) => {
    let url = '#unsupported-link-type';
    // Checks the content type of the linked content item
    if (link.type === 'article')
      url = `/articles/${link.urlSlug}`;
    return {
      linkHtml: `<a class="xLink" data-item-id="${linkId}" title="${linkText}" >${url}</a>`,
      // You can also return a plain URL
      linkUrl: url,
    };
});

// Gets the resolved HTML content of your rich text
const resolvedRichTextHtml = resolvedRichText.html;
```

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>
```

What's next?

- 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.