Register custom components

Want a shortcut? Get going right away with Builder's Devtools, where you can register components with the flip of a switch. For details, visit Using Builder Devtools for Automated Integration.

To use your custom components in the Builder Visual Editor, you need to take a couple of minimal steps in your codebase.

Prerequisites

To get the most out of this tutorial, you should have the following:

- Integrated Pages or Sections
- The Builder SDK installed for your framework ***except the HTML API***.

Setting up your code

To use your custom component with Builder, you need to let Builder know about it by registering your component. Registering your component makes the component appear in the Visual Editor as a custom block. Follow the steps according to framework.

Framework: React

Meta-Framework: Next.js (Pages Router)

SDK Generation: Gen 1

For more information on SSR and custom components, visit Builder's SSR and SSG documentation.

Registering your component

Register your component as in the following example. Here, `registerComponent()` uses Next.js `dynamic()` to register a `Heading` component with two inputs, `name` and `type`, and an icon.

```
// components/heading.tsx
const Heading = props => (
  <h1 className="my-heading">{props.title}</h1>
)

export default Heading;
```

```
// builder-registry.ts
import { Builder } from '@builder.io/react'

Builder.registerComponent(
  dynamic(() => import('./components/heading')),
  {
    name: 'Heading',
    inputs: [{ name: 'title', type: 'text' }],
    image: 'https://cdn.builder.io/api/v1/image/assets%2FYJIGb4i01jvw0SRdL
  }
)
```

The `options` you provide to `registerComponent()` determine how your component appears in the Visual Editor. For example, with the `image` option, you can provide your own icon that the Visual Editor uses in the **Insert** tab. For a list of additional options, refer to Input types.

As a best practice:

For a *dynamically* registered component example, see the Builder Blog Example on GitHub.

Tip: If you have a prop you'd like to make editable in the Visual Editor, be sure to define a input for it. In this example, `name` is editable because it is specified in the `inputs` array of `registerComponent()`.

Check out the full reference docs for `registerComponent()` options for more details.

Using your component in your code

After registering your component with Builder, you can use it as in this example where an `h1` uses `title` from the custom component, `Heading`.

Make sure that you import the file where `Builder.registerComponent()` resides into any file that has a `BuilderComponent`. **This import is vital to the setup.**

```
// page.tsx
import { BuilderComponent } from '@builder.io/react'

// IMPORTANT: import the file that you call Builder.registerComponent
// anywhere you have <BuilderComponent />
import '../builder-registry'

export default function Page() {
  return <BuilderComponent ... />
}
```

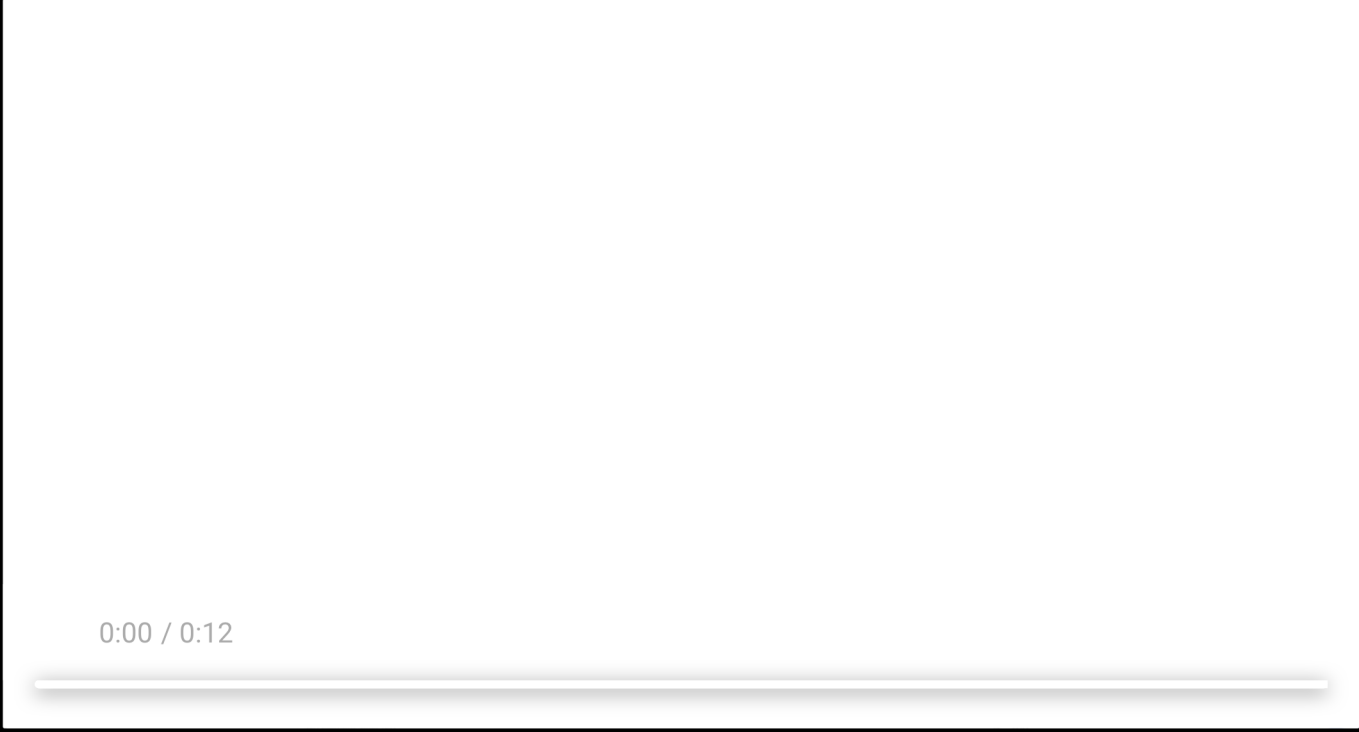
Your registered components, though available for use in the Visual Editor, reside in your code. **Builder does not retain or store your components in any way.**

Using your component in Builder's Visual Editor

The following video shows the **Custom Thing** custom component in the **Custom Components** section of the **Insert** tab.

`CustomThing` has an `input` registered with Builder called `name`. To use a registered component's input, do the following:

1. Drag and drop your custom component into the work area.
2. Select your component by clicking on it.
3. Click the **Edit** button.
4. Add a value for the the `input`, in this example, the `input` is called `name`, and the value added is the text, "Sunshine".



Setting component options

When you register a component, you must provide a `name`. Optionally, you can also specify:

- component default styling
- whether Builder wraps the component in a `<div>`
- whether the component displays in the **Insert** tab

`defaultStyles`	`object`	Use for showing an example value in the input form wh instance of this component, to users understand its pur Visual Editor can edit these styles. `Builder.registerComponent(HelloWorldCompon ... defaultStyles: { textAlign: "center", fontSize: "20px", }, ... });`
`hideFromInsertMenu`	`Boolean`	Hide your component from the **Insert** tab within Builder this feature for deprecating a component when you wo but still need the component registered with Builder so that uses the component continues to function properl information on versioning, refer to Versioning Custom C `hideFromInsertMenu: true`
`noWrap`	`Boolean`	By default, Builder wraps your components in a `<div>`. You can opt out of this wrapping by using the `noWrap` code example, see Builder's built-in form input compon When using `noWrap: true`, it is important to pass `{...` to ensure class names are assigned correctly as in the The following example shows `noWrap` with a Material U Builder.io. `import { TextField } from '@material-ui/co export const BuilderTextField = props => ( // Important! Builder.io must add a coup`

The following code snippet features a number of options in the context of `registerComponent()`:

```
const HeloWorldComponent = (props) => <div>{props.text}</div>;

Builder.registerComponent(HelloWorldComponent, {
  // Begin component options
  // Name your component (required)
  name: "Hello World",
  // Optional: provide CSS in an object to defaultStyles
  defaultStyles: {
    textAlign: "center",
    fontSize: "20px",
  },
  // Optional: specify whether your component is wrapped in a <div>
  noWrap: true,
  // Optional: specify if your component shows in the Insert tab
  hideFromInsertMenu: false,
  // End component options

  // Begin inputs:
  inputs: [{ ... }],
});
```

For more detail on all available `registerComponent()` options, read registerComponent Options.

multiple bindings simultaneously. This is helpful when, for example, applying styles from an existing design system. The snippet below registers a custom component with default bindings for the product title and a theme background color:

```
Builder.registerComponent(MyComponent, {
   name: 'MyComponent',
   defaults: {
     bindings: {
       'component.options.title': 'state.product.title',
       'style.background': 'state.theme.colors.primary'
     },
   },
 },
```

Fetch data async

Unless you're using a React-based framework, your components can fetch data async out-of-the-box.

If you are using a React-based framework and want your component to fetch data async, use Builder's `getAsyncProps` helper.

In the following example, the page loads after the all the data arrives:

```
export async function getStaticProps(context) {
  const content = await builder.get('page', { url: context.resolvedUrl }).
  await getAsyncProps(content, {
    async Products(props) {
      return {
        data: await fetch(`${apiRoot}/products?cat=${props.category}`).ther
      };
    },
  });
  return { props: { content } };
}
```

Make a component available only in specified models

When you use `Builder.registerComponent()` to register custom components in Builder.io, you can provide various options to customize the behavior of your component.

One of these options is the `models` property, where you can specify a list of model names. By doing so, you can restrict the usage of the component to only those models listed, ensuring that the component is available only for specific purposes. If the models property is not provided, the component will be available for all models in Builder.io.

To control exactly which models your component is available in, use the models array when registering your component as in the following example:

```
Builder.registerComponent(RelatedProducts, {
  name: 'RelatedProducts',
  models: ['product-editorial', 'page'],
  ....
})
```

In this example, the `RelatedProducts` component is only accessible in the Visual Editor when editing content entries made with the `product-editorial` or `page` models.

For more detail on all available `registerComponent()` options, read registerComponent Options.

What's next