Component Registration

> This page assumes you've already read the Components Basics. Read that first if you are new to components.

Watch a free video lesson on Vue School

A Vue component needs to be "registered" so that Vue knows where to locate its implementation when it is encountered in a template. There are two ways to register components: global and local.

Global Registration

We can make components available globally in the current Vue application using the `.component()` method:

```js
import { createApp } from 'vue'

const app = createApp({})

app.component(
  // the registered name
  'MyComponent',
  // the implementation
  {
    /* ... */
  }
)
```

If using SFCs, you will be registering the imported `.vue` files:

```js
import MyComponent from './App.vue'

app.component('MyComponent', MyComponent)
```

```js
app
  .component('ComponentA', ComponentA)
  .component('ComponentB', ComponentB)
  .component('ComponentC', ComponentC)
```

Globally registered components can be used in the template of any component within this application:

```template
<!-- this will work in any component inside the app -->
<ComponentA/>
<ComponentB/>
<ComponentC/>
```

This even applies to all subcomponents, meaning all three of these components will also be available *inside each other*.

Local Registration

While convenient, global registration has a few drawbacks:

1. Global registration prevents build systems from removing unused components (a.k.a "tree-shaking"). If you globally register a component but end up not using it anywhere in your app, it will still be included in the final bundle.
2. Global registration makes dependency relationships less explicit in large applications. It makes it difficult to locate a child component's implementation from a parent component using it. This can affect long-term maintainability similar to using too many global variables.

Local registration scopes the availability of the registered components to the current component only. It makes the dependency relationship more explicit, and is more tree-shaking friendly.

When using SFC with `<script setup>`, imported components can be locally used without registration:

```vue
<script setup>
import ComponentA from './ComponentA.vue'
</script>

<template>
```

In non-`<script setup>`, you will need to use the `components` option:

```js
import ComponentA from './ComponentA.js'

export default {
  components: {
    ComponentA
  },
  setup() {
    // ...
  }
}
```

For each property in the `components` object, the key will be the registered name of the component, while the value will contain the implementation of the component. The above example is using the ES2015 property shorthand and is equivalent to:

```js
export default {
  components: {
    ComponentA: ComponentA
  }
  // ...
}
```

Note that **locally registered components are *not* also available in descendant components**. In this case, `ComponentA` will be made available to the current component only, not any of its child or descendant components.

Component Name Casing

Throughout the guide, we are using PascalCase names when registering components. This is because:

1. PascalCase names are valid JavaScript identifiers. This makes it easier to import and register components in JavaScript. It also helps IDEs with auto-completion.

2. `<PascalCase />` makes it more obvious that this is a Vue component instead of a native HTML element in templates. It also differentiates Vue components from custom elements (web components).

This is the recommended style when working with SFC or string templates. However, as discussed in in-DOM Template Parsing Caveats, PascalCase tags are not usable in in-DOM templates.

a Vue template (or inside an HTML element rendered by Vue) via both `<MyComponent>` and `<my-component>`. This allows us to use the same JavaScript component registration code regardless of template source.

Edit this page on GitHub

Previous
Lifecycle Hooks

Next
Props