# Component v-model

Watch an interactive video lesson on Scrimba

## Basic Usage

`v-model` can be used on a component to implement a two-way binding.

Starting in Vue 3.4, the recommended approach to achieve this is using the `defineModel()` macro:

Child.vue

```vue
<script setup>
const model = defineModel()

function update() {
  model.value++
}
</script>

<template>
  <div>Parent bound v-model is: {{ model }}</div>
  <button @click="update">Increment</button>
</template>
```

The parent can then bind a value with `v-model`:

Parent.vue

```template
<Child v-model="countModel" />
```

The value returned by `defineModel()` is a ref. It can be accessed and mutated like any other ref, except that it acts as a two-way binding between a parent value and a local one:

This means you can also bind this ref to a native input element with `v-model`, making it straightforward to wrap native input elements while providing the same `v-model` usage:

```vue
<script setup>
const model = defineModel()
</script>

<template>
  <input v-model="model" />
</template>
```

Try it in the playground

### Under the Hood

`defineModel` is a convenience macro. The compiler expands it to the following:

- A prop named `modelValue`, which the local ref's value is synced with;
- An event named `update:modelValue`, which is emitted when the local ref's value is mutated.

This is how you would implement the same child component shown above prior to 3.4:

Child.vue

```vue
<script setup>
const props = defineProps(['modelValue'])
const emit = defineEmits(['update:modelValue'])
</script>

<template>
  <input
    :value="props.modelValue"
    @input="emit('update:modelValue', $event.target.value)"
  />
</template>
```

Then, `v-model="foo"` in the parent component will be compiled to:

Parent.vue

```template
<Child
  :modelValue="foo"
  @update:modelValue="$event => (foo = $event)"
/>
```

Because `defineModel` declares a prop, you can therefore declare the underlying prop's options by passing it to `defineModel`:

```js
// making the v-model required
const model = defineModel({ required: true })

// providing a default value
const model = defineModel({ default: 0 })
```

> ⚠ WARNING
>
> If you have a `default` value for `defineModel` prop and you don't provide any value for this prop from the parent component, it can cause a de-synchronization between parent and child components. In the example below, the parent's `myRef` is undefined, but the child's `model` is 1:
>
> Child.vue
>
> ```vue
> <script setup>
> const model = defineModel({ default: 1 })
> </script>
> ```
>
> Parent.vue
>
> ```vue
> <script setup>
> const myRef = ref()
> </script>
>
> <template>
>   <Child v-model="myRef"></Child>
> </template>
> ```

## `v-model` Arguments

`v-model` on a component can also accept an argument:

```template
<MyComponent v-model:title="bookTitle" />
```

MyComponent.vue

```vue
<script setup>
const title = defineModel('title')
</script>

<template>
  <input type="text" v-model="title" />
</template>
```

Try it in the Playground

If prop options are also needed, they should be passed after the model name:

```js
const title = defineModel('title', { required: true })
```

▶ Pre 3.4 Usage

## Multiple `v-model` Bindings

By leveraging the ability to target a particular prop and event as we learned before with `v-model` arguments, we can now create multiple `v-model` bindings on a single component instance.

Each `v-model` will sync to a different prop, without the need for extra options in the component:

```template
<UserName
  v-model:first-name="first"
  v-model:last-name="last"
/>
```

```vue
<script setup>
const firstName = defineModel('firstName')
const lastName = defineModel('lastName')
</script>

<template>
  <input type="text" v-model="firstName" />
  <input type="text" v-model="lastName" />
</template>
```

▶ Pre 3.4 Usage

## Handling `v-model` Modifiers

When we were learning about form input bindings, we saw that `v-model` has built-in modifiers - `.trim`, `.number` and `.lazy`. In some cases, you might also want the `v-model` on your custom input component to support custom modifiers.

Let's create an example custom modifier, `capitalize`, that capitalizes the first letter of the string provided by the `v-model` binding:

```template
<MyComponent v-model.capitalize="myText" />
```

Modifiers added to a component `v-model` can be accessed in the child component by destructuring the `defineModel()` return value like this:

```vue
<script setup>
const [model, modifiers] = defineModel()

console.log(modifiers) // { capitalize: true }
</script>

<template>
  <input type="text" v-model="model" />
</template>
```

To conditionally adjust how the value should be read / written based on modifiers, we can pass `get` and `set` options to `defineModel()`. These two options receive the value on get / set of the model ref and should return a transformed value. This is how we can use the `set` option to implement the `capitalize` modifier:

```vue
<script setup>
const [model, modifiers] = defineModel({
  set(value) {
    if (modifiers.capitalize) {
      return value.charAt(0).toUpperCase() + value.slice(1)
    }
    return value
  }
})
</script>
```

```
</template>
```

Try it in the Playground

▶ Pre 3.4 Usage

## Modifiers for `v-model` with Arguments

Here's another example of using modifiers with multiple `v-model` with different arguments:

```template
<UserName
  v-model:first-name.capitalize="first"
  v-model:last-name.uppercase="last"
/>
```

```vue
<script setup>
const [firstName, firstNameModifiers] = defineModel('firstName')
const [lastName, lastNameModifiers] = defineModel('lastName')

console.log(firstNameModifiers) // { capitalize: true }
console.log(lastNameModifiers) // { uppercase: true }
</script>
```

▶ Pre 3.4 Usage

Edit this page on GitHub