Vue 3 with TypeScript cheat sheet

Back up

How to properly use types when writing Vue components.

Published **29 July 2022** Last updated **Mar 2023**

frontend typescript vue 3

Just a quick cheat sheet on how to define and do basic stuff when using the **composition API** with **TypeScript**.

Table of contents

Prerequisites

For this, you'll need a **Vue 3 + TypeScript (+ Tailwind CSS)** project.
You can set up one following the instructions here:
Build a Vue 3 + TypeScript dev environment with Vite

Basic types, Records

- If you want a type meaning "any value", you probably want `unknown` instead.
- If you want a type meaning "any object", you probably want `Record<string, unknown>` instead.

- If you want a type meaning "empty object", you probably want `Record<string, never>` instead.

Adding properties to the `window` object

Add a `src/index.d.ts` file with this content:

```
export {}

declare global {
  interface Window {
    someVariable: string
    otherThing: number
    // any other variables you need here...
  }
}
```

With that you'll avoid the error:

> *"Property 'someVariable' does not exist on type 'Window & typeof globalThis'."*

Vue components

If you are passing a Vue component as a property or assigning to a variable:

```
import { defineComponent } from 'vue'

export interface MenuItem {
  label: string
  icon?: ReturnType<typeof defineComponent>
  children: MenuItem[]
}
```

Props

```
<script setup lang="ts">
const props = defineProps<{
  ppi: number | null
  mapConfig: MapConfig
}>()
</script>
```

Props with defaults

```
<script setup lang="ts">
const props = withDefaults(
  defineProps<{
    buttonStyle?: 'primary' | 'secondary'
  }>(),
  { buttonStyle: 'primary' }
)
</script>
```

Emits

```
<script setup lang="ts">
const emit = defineEmits<{
  (e: 'frame:height', value: number): void
  (e: 'frame:width', value: number): void
  (e: 'layer:toggle', value: LayerSpecification): void
  (e: 'map:add-text', value: string): void
  (e: 'map:download'): void
  (e: 'text:remove', value: [AddedText, number]): void
}>()
</script>
```

Refs

Typing refs:

```
<script setup lang="ts">
```

```
import { ref } from 'vue'

const activeMenuIndex = ref<number | null>(null)

// later...
activeMenuIndex.value = 5
activeMenuIndex.value = null
</script>
```

DOM refs

Get a DOM reference to an HTML input element:

```
<script setup lang="ts">
import { ref } from 'vue'

const textToAdd = ref('')
const textToAddInput = ref<HTMLInputElement | null>(null)

onMounted(() => {
  console.log(textToAddInput.value)
})

function focusInput() {
  textToAddInput.value?.focus()
}
</script>

<template>
  <input
    ref="textToAddInput"
    v-model.trim="textToAdd"
    type="text"
  />
</template>
```

Component refs

Get a DOM reference to a Vue component:

```
<script setup lang="ts">
import { ref } from 'vue'

import BaseButton from '@/components/buttons/BaseButton.vue'

const buttonRef = ref<InstanceType<typeof BaseButton> | null>(null)

onMounted(() => {
  console.log(buttonRef.value)
})

</script>

<template>
  <BaseButton ref="buttonRef">Click me</BaseButton>
</template>
```

Events

When working with input events, the event handler will receive an `**Event**` type, then you'll have to assert the `**currentTarget**` type as the one you need.

```
<script setup lang="ts">
function handleHeightResize(ev: Event) {
  const value = (ev.currentTarget as HTMLInputElement).value

  if (value !== '') {
    const inPixels = toPixels(parseInt(value))
    emit('frame:height', inPixels)
  }
}
</script>

<template>
  <input
    type="number"
    min="0"
```

```
    step="1"
    :value="frameHeightInUnits"
    @input="handleHeightResize"
  />
</template>
```

Provide / inject

The way I see it, `**provide**` and `**inject**` are basically localized global props.

```
// TheParent.vue
const ppi = ref<number | null>(null)
provide('ppi', ppi)

// AGrandGrandGrandChild.ts
const ppi = inject<Ref<number | null>>('ppi')
```

Watch

Watching some values that in turn will update another value that depends on a DOM element that changes based on the watched values (need to wait for next tick!):

```
<script setup lang="ts">
const boundingRect = ref<DOMRect | undefined>(undefined)
const height = computed(() => `${props.height}px`)
const width = computed(() => `${props.width}px`)

watch([height, width], () => {
  nextTick(() => {
    boundingRect.value = frameRef.value?.getBoundingClientRect()
  })
})

defineExpose({
  boundingRect
})
</script>
```

```
<template>
  <div
    ref="frameRef"
    class="frame"
  ></div>
</template>

<style scoped>
.frame {
  height: v-bind(height);
  width: v-bind(width);
}
</style>
```

Somewhere in the parent:

```
const frameRef = ref<{ boundingRect: DOMRect } | null>(null)

// later...
function handleMapDownload() {
  const boundingRect: DOMRect | undefined = frameRef.value?.boundingRect
```

setTimeout

```
type MaybeTimeout = ReturnType<typeof setTimeout> | undefined

let timeoutId: MaybeTimeout = undefined

function frequentlyCalled() {
  clearTimeout(timeoutId)

  // Do stuff...

  timeoutId = setTimeout(() => {
    // Do some other stuff on time out
  }, 500)
}
```

setInterval

What about **setInterval?**

```
type MaybeInterval = ReturnType<typeof setInterval> | undefined

let intervalId: MaybeInterval = undefined

onMounted(() => {
  intervalId = setInterval(() => console.log("I'm called every 3 seconds..."), 3
})

onBeforeUnmount(() => {
  if (intervalId) {
    clearInterval(intervalId)
  }
})
```

Silence an error

If you need to quickly —and hopefully temporarily— silence a **TypeScript** error you can do so with:

```
// @ts-expect-error whatever reason here
const algo: any
```

Recursive types

Useful when defining tree-like structures.
Seems they can only be used when defining the types of properties.

```
interface SomeTree {
  [x: string]: boolean | SomeTree
}
```

Functions

What about a function that can receive a function as an argument or nothing at all?

```
function toggleModal(hideMenu: (() => void) | void)
```

References

- Template Refs - Accessing the Refs
- <script setup> - defineExpose()
- TypeScript with Composition API
- Watchers - Basic Example
- Provide / Inject
- <script setup> - props withDefaults()

Got comments or feedback?

Follow me on

v-57f6bb4