Introduction to the Script Setup Syntax

While the evolution from Vue 3.1 to 3.2 was technically a *minor* version update, it comes with a *major* improvement in TypeScript experience. This new version of Vue promotes the *Script Setup* syntax, which was previously an experimental feature, and is now an **officially supported** one. With it, we have a simpler way to write components with the Composition API and a more elegant way of using TypeScript in our Vue apps.

Composition API (original)

```
<script>
import { ref } from 'vue'

export default {
  setup() {
    const title = ref('Hello')
    return { title }
  }
}
</script>
```

Composition API (with Script Setup)

```
<script setup>
import { ref } from 'vue'


const title = ref('Hello')


</script>
```

> While the above snippets are in pure JavaScript, this simplicity carries over to the TypeScript side of Vue as well.

This tutorial will focus on the TypeScript aspects of using the new *Script Setup* syntax. So to get the most out of this tutorial, make sure that you have a basic familiarity with *TypeScript*, the *Composition API*, and the *Script Setup* syntax. If you need a refresher on any of these, you can check out the official docs or the Vue Mastery courses on these subjects.

TypeScript with Script Setup

Although the *Script Setup* syntax is just syntactic sugar for the Composition API, it gives us highly useful features to use when working with props and emits in TypeScript. Since the *Script Setup* syntax is still just the Composition API under the hood, we will still be writing most of our TypeScript code the same way as with the `setup()` function, such as creating `ref`, `reactive`, and functions. But the way that we'll be using props and emit is very different in this new syntax.

So first, we'll go through some TypeScript fundamentals for working with the Composition API, and then we'll get into the new TypeScript features of the *Script Setup* syntax. The goal is to provide a well-rounded tour of the Vue-TypeScript combination that features the newest recommended practices.

Throughout the article, we'll be covering the following topics:

- ref
- reactive
- type inference
- functions
- props
- emit

Setup

If you want to follow along with the code in this tutorial, you can create a new Vue TypeScript app using Vite. If you're new to Vite, you can check out our course taught by its creator, Evan You.

```
npm init vite@latest my-app -- --template vue-ts
cd my-app
npm install
npm run dev
```

As a newer alternative, you can also use the official scaffolding tool `create-vue`:

```
npm init vue@latest
```

To opt-in to using the *Script Setup* syntax, you can simply just add the `setup` attribute to the `script` tag:

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

</script>
```

You can opt-in this feature in any one of your components.

IDE Support

For the best IDE support, you can use *VSCode* with the *Volar* plugin. As an alternative, you can also use *WebStorm*, which comes with built-in support for *Script Setup*.

So when you put your mouse cursor on a variable, it will show you its type:

```
  aler const props: PropsWithDefaults<Readonly<Props>, {
}         alertMessageOnLimit: "can not go any higher";
        }>
const props = withDefaults(defineProps<Props>(), {
  alertMessageOnLimit: 'can not go any higher'
})
```

Then same when you do that in the `<template>`:

```
<templat var count: number | null
  <p>{{ count }}</p>
  <AddButton
    @add-count="addCount"
```

Now let's begin our tour of the new Vue TypeScript experience. We'll start with the most fundamental of all things: creating reactive variables.

Reactive Variables with Type Inference

Typing Your Callback Functions

Unlock this lesson by subscribing to a plan.

Unlock Content

Props with Compiler Macros

In this lesson, we'll look at the most important change in the script setup syntax,

Type-Safe Emit

Just like props, emit has a total makeover in the script setup syntax. In this

(Bonus) Vue vs React: Framework Philosophy

Unlock this lesson by subscribing to a plan.

Unlock Content