API Reference

#<router-link>

<router-link> is the component for enabling user navigation in a router-enabled app. The target location is specified with the to prop. It renders as an <a> tag with correct href by default, but can be configured with the tag prop. In addition, the link automatically gets an active CSS class when the target route is active.

<router-link> is preferred over hard-coded <a href="..."> for the following reasons:

It works the same way in both HTML5 history mode and hash mode, so if you ever decide to switch mode, or when the router falls back to hash mode in IE9, nothing needs to be changed.

In HTML5 history mode, router-link will intercept the click event so that the browser doesn't try to reload the page.

When you are using the base option in HTML5 history mode, you don't need to include it in to prop's URLs.

#v-slot API (3.1.0+)

router-link exposes a low level customization through a scoped slot (opens new window). This is a more advanced API that primarily targets library authors but can come in handy for developers as well, most of the time in a custom component like a NavLink or other.

When using the v-slot API, it is required to pass one single child to router-link. If you don't, router-link will wrap its children in a span element.

<router-link

to="/about"

custom

v-slot="{ href, route, navigate, isActive, isExactActive }"

>

<NavLink :active="isActive" :href="href" @click="navigate"

>{{ route.fullPath }}</NavLink

>

</router-link>

href: resolved url. This would be the href attribute of an a element

route: resolved normalized location

navigate: function to trigger the navigation. It will automatically prevent events when necessary, the same way router-link does

isActive: true if the active class should be applied. Allows to apply an arbitrary class

isExactActive: true if the exact active class should be applied. Allows to apply an arbitrary class

#Example: Applying Active Class to Outer Element

Sometimes we may want the active class to be applied to an outer element rather than the <a> tag itself, in that case, you can wrap that element inside a router-link and use the v-slot properties to create your link:

<router-link

to="/foo"

v-slot="{ href, route, navigate, isActive, isExactActive }"

custom

>

<li

:class="[isActive && 'router-link-active', isExactActive && 'router-link-exact-active']"

>

<a :href="href" @click="navigate">{{ route.fullPath }}</a>

</li>

</router-link>

TIP

If you add a target="\_blank" to your a element, you must omit the @click="navigate" handler.

#<router-link> Props

#to

type: string | Location

required

Denotes the target route of the link. When clicked, the value of the to prop will be passed to router.push() internally, so the value can be either a string or a location descriptor object.

<!-- literal string -->

<router-link to="home">Home</router-link>

<!-- renders to -->

<a href="home">Home</a>

<!-- javascript expression using `v-bind` -->

<router-link v-bind:to="'home'">Home</router-link>

<!-- Omitting `v-bind` is fine, just as binding any other prop -->

<router-link :to="'home'">Home</router-link>

<!-- same as above -->

<router-link :to="{ path: 'home' }">Home</router-link>

<!-- named route -->

<router-link :to="{ name: 'user', params: { userId: 123 }}">User</router-link>

<!-- with query, resulting in `/register?plan=private` -->

<router-link :to="{ path: 'register', query: { plan: 'private' }}"

>Register</router-link

>

#replace

type: boolean

default: false

Setting replace prop will call router.replace() instead of router.push() when clicked, so the navigation will not leave a history record.

<router-link :to="{ path: '/abc'}" replace></router-link>

#append

type: boolean

default: false

Setting append prop always appends the relative path to the current path. For example, assuming we are navigating from /a to a relative link b, without append we will end up at /b, but with append we will end up at /a/b.

<router-link :to="{ path: 'relative/path'}" append></router-link>

#tag

type: string

default: "a"

Sometimes we want <router-link> to render as another tag, e.g <li>. Then we can use tag prop to specify which tag to render to, and it will still listen to click events for navigation.

<router-link to="/foo" tag="li">foo</router-link>

<!-- renders as -->

<li>foo</li>

#active-class

type: string

default: "router-link-active"

Configure the active CSS class applied when the link is active. Note the default value can also be configured globally via the linkActiveClass router constructor option.

#exact

type: boolean

default: false

The default active class matching behavior is inclusive match. For example, <router-link to="/a"> will get this class applied as long as the current path starts with /a/ or is /a.

One consequence of this is that <router-link to="/"> will be active for every route! To force the link into "exact match mode", use the exact prop:

<!-- this link will only be active at `/` -->

<router-link to="/" exact></router-link>

Check out more examples explaining active link class live (opens new window).

#exact-path

New in 3.5.0

type: boolean

default: false

Allows matching only using the path section of the url, effectively ignoring the query and the hash sections.

<!-- this link will also be active at `/search?page=2` or `/search#filters` -->

<router-link to="/search" exact-path> </router-link>

#exact-path-active-class

type: string

default: "router-link-exact-path-active"

Configure the active CSS class applied when the link is active with exact path match. Note the default value can also be configured globally via the linkExactPathActiveClass router constructor option.

#event

type: string | Array<string>

default: 'click'

Specify the event(s) that can trigger the link navigation.

#exact-active-class

type: string

default: "router-link-exact-active"

Configure the active CSS class applied when the link is active with exact match. Note the default value can also be configured globally via the linkExactActiveClass router constructor option.

#aria-current-value

type: 'page' | 'step' | 'location' | 'date' | 'time' | 'true' | 'false'

default: "page"

Configure the value of aria-current when the link is active with exact match. It must be one of the allowed values for aria-current (opens new window)in the ARIA spec. In most cases, the default of page should be the best fit.

#<router-view>

The <router-view> component is a functional component that renders the matched component for the given path. Components rendered in <router-view> can also contain their own <router-view>, which will render components for nested paths.

Any non-name props will be passed along to the rendered component, however most of the time the per-route data is contained in the route's params.

Since it's just a component, it works with <transition> and <keep-alive>. When using them both together, make sure to use <keep-alive> inside:

<transition>

<keep-alive>

<router-view></router-view>

</keep-alive>

</transition>

#<router-view> Props

#name

type: string

default: "default"

When a <router-view> has a name, it will render the component with the corresponding name in the matched route record's components option. See Named Views for an example.

#Router Construction Options

#routes

type: Array<RouteConfig>

Type declaration for RouteConfig:

interface RouteConfig = {

path: string,

component?: Component,

name?: string, // for named routes

components?: { [name: string]: Component }, // for named views

redirect?: string | Location | Function,

props?: boolean | Object | Function,

alias?: string | Array<string>,

children?: Array<RouteConfig>, // for nested routes

beforeEnter?: (to: Route, from: Route, next: Function) => void,

meta?: any,

// 2.6.0+

caseSensitive?: boolean, // use case sensitive match? (default: false)

pathToRegexpOptions?: Object // path-to-regexp options for compiling regex

}

#mode

type: string

default: "hash" (in browser) | "abstract" (in Node.js)

available values: "hash" | "history" | "abstract"

Configure the router mode.

hash: uses the URL hash for routing. Works in all Vue-supported browsers, including those that do not support HTML5 History API.

history: requires HTML5 History API and server config. See HTML5 History Mode.

abstract: works in all JavaScript environments, e.g. server-side with Node.js. The router will automatically be forced into this mode if no browser API is present.

#base

type: string

default: "/"

The base URL of the app. For example, if the entire single page application is served under /app/, then base should use the value "/app/".

#linkActiveClass

type: string

default: "router-link-active"

Globally configure <router-link> default active class. Also see router-link.

#linkExactActiveClass

type: string

default: "router-link-exact-active"

Globally configure <router-link> default active class for exact matches. Also see router-link.

#scrollBehavior

type: Function

Signature:

type PositionDescriptor =

{ x: number, y: number } |

{ selector: string } |

void

type scrollBehaviorHandler = (

to: Route,

from: Route,

savedPosition?: { x: number, y: number }

) => PositionDescriptor | Promise<PositionDescriptor>

For more details see Scroll Behavior.

#parseQuery / stringifyQuery

type: Function

Provide custom query string parse / stringify functions. Overrides the default.

#fallback

type: boolean

default: true

Controls whether the router should fallback to hash mode when the browser does not support history.pushState but mode is set to history.

Setting this to false essentially makes every router-link navigation a full page refresh in IE9. This is useful when the app is server-rendered and needs to work in IE9, because a hash mode URL does not work with SSR.

#Router Instance Properties

#router.app

type: Vue instance

The root Vue instance the router was injected into.

#router.mode

type: string

The mode the router is using.

#router.currentRoute

type: Route

The current route represented as a Route Object.

#router.START\_LOCATION (3.5.0+)

type: Route

Initial route location represented as a Route Object where the router starts at. Can be used in navigation guards to differentiate the initial navigation.

import VueRouter from 'vue-router'

const router = new VueRouter({

// ...

})

router.beforeEach((to, from) => {

if (from === VueRouter.START\_LOCATION) {

// initial navigation

}

})

#Router Instance Methods

#router.beforeEach

#router.beforeResolve

#router.afterEach

Signatures:

router.beforeEach((to, from, next) => {

/\* must call `next` \*/

})

router.beforeResolve((to, from, next) => {

/\* must call `next` \*/

})

router.afterEach((to, from) => {})

Add global navigation guards. See Navigation Guards for more details.

All three methods return a function that removes the registered guard/hook.

#router.push

#router.replace

#router.go

#router.back

#router.forward

Signatures:

router.push(location, onComplete?, onAbort?)

router.push(location).then(onComplete).catch(onAbort)

router.replace(location, onComplete?, onAbort?)

router.replace(location).then(onComplete).catch(onAbort)

router.go(n)

router.back()

router.forward()

Programmatically navigate to a new URL. See Programmatic Navigation for more details.

These functions can only be called after installing the Router plugin and passing it to the root Vue instance as shown in the Getting Started.

#router.getMatchedComponents

Signature:

const matchedComponents: Array<Component> = router.getMatchedComponents(location?)

Returns an Array of the components (definition/constructor, not instances) matched by the provided location or the current route. This is mostly used during server-side rendering to perform data prefetching.

#router.resolve

Signature:

const resolved: {

location: Location;

route: Route;

href: string;

} = router.resolve(location, current?, append?)

Reverse URL resolving. Given location in form same as used in <router-link/>.

current is the current Route by default (most of the time you don't need to change this)

append allows you to append the path to the current route (as with router-link)

#router.addRoutes

DEPRECATED: use router.addRoute() instead.

Signature:

router.addRoutes(routes: Array<RouteConfig>)

Dynamically add more routes to the router. The argument must be an Array using the same route config format with the routes constructor option.

#router.addRoute

New in 3.5.0

Add a new route to the router. If the route has a name and there is already an existing one with the same one, it overwrites it.

Signature:

addRoute(route: RouteConfig): () => void

#router.addRoute

New in 3.5.0

Add a new route record as the child of an existing route. If the route has a name and there is already an existing one with the same one, it overwrites it.

Signature:

addRoute(parentName: string, route: RouteConfig): () => void

#router.getRoutes

New in 3.5.0

Get the list of all the active route records. Note only documented properties are considered Public API, avoid using any other property e.g. regex as it doesn't exist on Vue Router 4.

Signature:

getRoutes(): RouteRecord[]

#router.onReady

Signature:

router.onReady(callback, [errorCallback])

This method queues a callback to be called when the router has completed the initial navigation, which means it has resolved all async enter hooks and async components that are associated with the initial route.

This is useful in server-side rendering to ensure consistent output on both the server and the client.

The second argument errorCallback is only supported in 2.4+. It will be called when the initial route resolution runs into an error (e.g. failed to resolve an async component).

#router.onError

Signature:

router.onError(callback)

Register a callback which will be called when an error is caught during a route navigation. Note for an error to be called, it must be one of the following scenarios:

The error is thrown synchronously inside a route guard function;

The error is caught and asynchronously handled by calling next(err) inside a route guard function;

An error occurred when trying to resolve an async component that is required to render a route.

#The Route Object

A route object represents the state of the current active route. It contains parsed information of the current URL and the route records matched by the URL.

The route object is immutable. Every successful navigation will result in a fresh route object.

The route object can be found in multiple places:

Inside components as this.$route

Inside $route watcher callbacks

As the return value of calling router.match(location)

Inside navigation guards as the first two arguments:

router.beforeEach((to, from, next) => {

// `to` and `from` are both route objects

})

Inside the scrollBehavior function as the first two arguments:

const router = new VueRouter({

scrollBehavior(to, from, savedPosition) {

// `to` and `from` are both route objects

}

})

#Route Object Properties

$route.path

type: string

A string that equals the path of the current route, always resolved as an absolute path. e.g. "/foo/bar".

$route.params

type: Object

An object that contains key/value pairs of dynamic segments and star segments. If there are no params the value will be an empty object.

$route.query

type: Object

An object that contains key/value pairs of the query string. For example, for a path /foo?user=1, we get $route.query.user == 1. If there is no query the value will be an empty object.

$route.meta

type: Object

An object that contains key/value pairs of the route meta object. If there are no meta properties the value will be an empty object.

$route.hash

type: string

The hash of the current route (with the #), if it has one. If no hash is present the value will be an empty string.

$route.fullPath

type: string

The full resolved URL including query and hash.

$route.matched

type: Array<RouteRecord>

An Array containing route records for all nested path segments of the current route. Route records are the copies of the objects in the routes configuration Array (and in children Arrays):

const router = new VueRouter({

routes: [

// the following object is a route record

{

path: '/foo',

component: Foo,

children: [

// this is also a route record

{ path: 'bar', component: Bar }

]

}

]

})

When the URL is /foo/bar, $route.matched will be an Array containing both objects (cloned), in parent to child order.

$route.name

The name of the current route, if it has one. (See Named Routes)

$route.redirectedFrom

The name of the route being redirected from, if there were one. (See Redirect and Alias)

#Component Injections

#Component Injected Properties

These properties are injected into every child component by passing the router instance to the root instance as the router option.

this.$router

The router instance.

this.$route

The current active Route. This property is read-only and its properties are immutable, but it can be watched.

#Component Enabled Options

beforeRouteEnter

beforeRouteUpdate

beforeRouteLeave

See In Component Guards.