API Reference

Vuex.Store

```js
import Vuex from 'vuex'

const store = new Vuex.Store({ ...options })
```

Vuex.Store Constructor Options

- **state**
  - type: `Object`

    The root state object for the Vuex store.

- **mutations**
  - type: `{ [type: string]: Function }`

    Register mutations on the store. The handler function always receives `state` as the first argument (will be module local state if defined in a module), and receives a second `payload` argument if there is one.

- **actions**
  - type: `{ [type: string]: Function }`

    Register actions on the store. The handler function receives a `context` object that exposes the following properties:

    ```js
    {
      state,     // same as store.state, or local state if in modules
      rootState, // same as store.state, only in modules
      commit,    // same as store.commit
      dispatch,  // same as store.dispatch
      getters    // same as store.getters
    }
    ```

- **getters**
  - type: `{ [key: string]: Function }`

    Register getters on the store. The getter function receives the following arguments:

    ```
    state,     // will be module local state if defined in a module.
    getters    // same as store.getters
    ```

    Specific when defined in a module

    ```
    state,       // will be module local state if defined in a module.
    getters,     // module local getters of the current module
    rootState,   // global state
    rootGetters  // all getters
    ```

    Registered getters are exposed on `store.getters`.

- **modules**
  - type: `Object`

    An object containing sub modules to be merged into the store, in the shape of:

    ```js
    {
      key: {
        state,
        namespaced?,
        mutations?,
        actions?,
        getters?,
        modules?
      },
      ...
    }
    ```

    Each module can contain `state` and `mutations` similar to the root options. A module's state will be attached to the store's root state using the module's key. A module's mutations and getters will only receives the module's local state as the first argument instead of the root state, and module actions' `context.state` will also point to the local state.

- **plugins**
  - type: `Array<Function>`

    An array of plugin functions to be applied to the store. The plugin simply receives the store as the only argument and can either listen to mutations (for outbound data persistence, logging, or debugging) or dispatch mutations (for inbound data e.g. websockets or observables).

- **strict**
  - type: `Boolean`
  - default: `false`

    Force the Vuex store into strict mode. In strict mode any mutations to Vuex state outside of mutation handlers will throw an Error.

Vuex.Store Instance Properties

- **state**
  - type: `Object`

    The root state. Read only.

- **getters**
  - type: `Object`

    Exposes registered getters. Read only.

Vuex.Store Instance Methods

- **`commit(type: string, payload?: any, options?: Object) | commit(mutation: Object, options?: Object)`**
- **`dispatch(type: string, payload?: any, options?: Object) | dispatch(action: Object, options?: Object)`**
- **`replaceState(state: Object)`**

  Replace the store's root state. Use this only for state hydration / time-travel purposes.

- **`watch(getter: Function, cb: Function, options?: Object)`**

  Reactively watch a getter function's return value, and call the callback when the value changes. The getter receives the store's state as the only argument. Accepts an optional options object that takes the same options as Vue's `vm.$watch` method.

  To stop watching, call the returned handle function.

- **`subscribe(handler: Function)`**

  Subscribe to store mutations. The `handler` is called after every mutation and receives the mutation descriptor and post-mutation state as arguments:

  ```js
  store.subscribe((mutation, state) => {
    console.log(mutation.type)
    console.log(mutation.payload)
  })
  ```

- **`registerModule(path: string | Array<string>, module: Module)`**

- **`unregisterModule(path: string | Array<string>)`**

- **`hotUpdate(newOptions: Object)`**

Component Binding Helpers

- **`mapState(namespace?: string, map: Array<string> | Object): Object`**

- **`mapGetters(namespace?: string, map: Array<string> | Object): Object`**

- **`mapActions(namespace?: string, map: Array<string> | Object): Object`**

- **`mapMutations(namespace?: string, map: Array<string> | Object): Object`**

- **`createNamespacedHelpers(namespace: string): Object`**