Firing Events

> **Note**
>
> Most projects have a few use cases for `fireEvent`, but the majority of the time you should probably use `@testing-library/user-event`.

`fireEvent`

```
fireEvent(node: HTMLElement, event: Event)
```

Fire DOM events.

```
// <button>Submit</button>
fireEvent(
```



```
    bubbles: true,
    cancelable: true,
  }),
)
```

`fireEvent[eventName]`

```
fireEvent[eventName](node: HTMLElement, eventProperties: Object)
```

Convenience methods for firing DOM events. Check out src/event-map.js for a full list as well as default `eventProperties`.

target: When an event is dispatched on an element, the event has the subjected element on a property called `target`. As a convenience, if you provide a `target` property in the `eventProperties` (second argument), then those properties will be assigned to the node which is receiving the event.

This is particularly useful for a change event:

```
fireEvent.change(getByLabelText(/username/i), {target: {value: 'a'}})

// note: attempting to manually set the files property of an HTMLInputEle
// results in an error as the files property is read-only.
// this feature works around that by using Object.defineProperty.
fireEvent.change(getByLabelText(/picture/i), {
  target: {
    files: [new File(['(⌐□_□)'], 'chucknorris.png', {type: 'image/png'})]
  },
})

// Note: The 'value' attribute must use ISO 8601 format when firing a
// change event on an input of type "date". Otherwise the element will no
// reflect the changed value.

// Invalid:
fireEvent.change(input, {target: {value: '24/05/2020'}})

// Valid:
fireEvent.change(input, {target: {value: '2020-05-24'}})
```

dataTransfer: Drag events have a `dataTransfer` property that contains data transferred during the operation. As a convenience, if you provide a `dataTransfer` property in the `eventProperties` (second argument), then those properties will be added to the event.

This should predominantly be used for testing drag and drop interactions.

```
fireEvent.drop(getByLabelText(/drop files here/i), {
  dataTransfer: {
    files: [new File(['(⌐□_□)'], 'chucknorris.png', {type: 'image/png'})]
  },
})
```

Keyboard events: There are three event types related to keyboard input - `keyPress`, `keyDown`, and `keyUp`. When firing these you need to reference an element in the DOM and the key you want to fire.

```
fireEvent.keyDown(domNode, {key: 'Enter', code: 'Enter'})

fireEvent.keyDown(domNode, {key: 'A', code: 'KeyA'})
```

You can find out which key code to use at https://keycode.info/.

`createEvent[eventName]`

```
createEvent[eventName](node: HTMLElement, eventProperties: Object)
```

Convenience methods for creating DOM events that can then be fired by `fireEvent`, allowing you to have a reference to the event created: this might be useful if you need to access event properties that cannot be initiated programmatically (such as `timeStamp`).

```
const myEvent = createEvent.click(node, {button: 2})
fireEvent(node, myEvent)
// myEvent.timeStamp can be accessed just like any other properties from r
// note: The access to the events created by `createEvent` is based on the
// Therefore, native properties of HTMLEvent object (e.g. `timeStamp`, `c
// For more info see: https://developer.mozilla.org/en-US/docs/Web/API/Eve
```

You can also create generic events:

```
// simulate the 'input' event on a file input
fireEvent(
  input,
  createEvent('input', input, {
    target: {files: inputFiles},
    ...init,
```

```
  }),
)
```

Using Jest Function Mocks

Jest's Mock functions can be used to test that a callback passed to the function was called, or what it was called when the event that **should** trigger the callback function does trigger the bound callback.

React

```
import {render, screen, fireEvent} from '@testing-library/react'

const Button = ({onClick, children}) => (
  <button onClick={onClick}>{children}</button>
)

test('calls onClick prop when clicked', () => {
  const handleClick = jest.fn()
  render(<Button onClick={handleClick}>Click Me</Button>)
  fireEvent.click(screen.getByText(/click me/i))
  expect(handleClick).toHaveBeenCalledTimes(1)
})
```

Edit this page

*Last updated on **7/22/2021** by **Nick McCurdy***

Previous
« ByTestId

Next
Async Methods »

fireEvent

> `fireEvent[eventName]`
>
> `createEvent[eventName]`
>
> Using Jest Function Mocks

Docs

Getting Started

Examples

API

Help

Community

Blog

Stack Overflow

Discord

More

☆ Star 15,107

GitHub

Edit Docs on GitHub

Hosted by Netlify