**Event Types**

**App Events**

These events come from the application currently under test (your application). These are the most useful events for you to listen to.

| **Event** | **Details** |
| --- | --- |
| **Name:** | uncaught:exception |
| **Yields:** | the error **(Object)**, Mocha runnable **(Object)** |
| **Description:** | Fires when an uncaught exception occurs in your application. Cypress will fail the test when this fires. Return false from this event and Cypress will not fail the test. Also useful for debugging purposes because the actual error instance is provided to you. See our recipe [Handling errors](https://docs.cypress.io/examples/recipes#Fundamentals). |

| **Event** | **Details** |
| --- | --- |
| **Name:** | window:confirm |
| **Yields:** | the confirmation text **(String)** |
| **Description:** | Fires when your app calls the global window.confirm() method. Cypress will auto accept confirmations. Return false from this event and the confirmation will be canceled. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | window:alert |
| **Yields:** | the alert text **(String)** |
| **Description:** | Fires when your app calls the global window.alert() method. Cypress will auto accept alerts. You cannot change this behavior. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | window:before:load  [End-to-End Only](https://docs.cypress.io/guides/core-concepts/testing-types" \l "What-is-E2E-Testing) |
| **Yields:** | the remote window **(Object)** |
| **Description:** | Fires as the page begins to load, but before any of your applications JavaScript has executed. This fires at the exact same time as cy.visit() onBeforeLoad callback. Useful to modify the window on a page transition. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | window:load |
| **Yields:** | the remote window **(Object)** |
| **Description:** | Fires after all your resources have finished loading after a page transition. This fires at the exact same time as a cy.visit() onLoad callback. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | window:before:unload  [End-to-End Only](https://docs.cypress.io/guides/core-concepts/testing-types" \l "What-is-E2E-Testing) |
| **Yields:** | the actual beforeunload event **(Object)** |
| **Description:** | Fires when your application is about to navigate away. The real event object is provided to you. Your app may have set a returnValue on the event, which is useful to assert on. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | window:unload  [End-to-End Only](https://docs.cypress.io/guides/core-concepts/testing-types" \l "What-is-E2E-Testing) |
| **Yields:** | the actual unload event **(Object)** |
| **Description:** | Fires when your application has unloaded and is navigating away. The real event object is provided to you. This event is not cancelable. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | url:changed |
| **Yields:** | the new url **(String)** |
| **Description:** | Fires whenever Cypress detects that your application's URL has changed. |

**Cypress Events**

These events come from Cypress as it issues commands and reacts to their state. These are all useful to listen to for debugging purposes.

| **Event** | **Details** |
| --- | --- |
| **Name:** | fail |
| **Yields:** | the error **(Object)**, Mocha runnable **(Object)** |
| **Description:** | Fires when the test has failed. Binding to this event without rethrowing the error will prevent the test from failing. However this is **strongly discouraged**. Tests should never legitimately fail. This event exists because it's extremely useful for debugging purposes. See our recipe [Handling errors](https://docs.cypress.io/examples/recipes#Fundamentals). |

| **Event** | **Details** |
| --- | --- |
| **Name:** | viewport:changed |
| **Yields:** | the new viewport **(Object)** |
| **Description:** | Fires whenever the viewport changes via a cy.viewport() or naturally when Cypress resets the viewport to the default between tests. Useful for debugging purposes. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | scrolled |
| **Yields:** | the element or window being scrolled **(Object)** |
| **Description:** | Fires whenever **Cypress** is scrolling your application. This event is fired when Cypress is [waiting for and calculating actionability](https://docs.cypress.io/guides/core-concepts/interacting-with-elements). It will scroll to 'uncover' elements currently being covered. This event is extremely useful to debug why Cypress may think an element is not interactive. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | command:enqueued |
| **Yields:** | command properties and arguments **(Object)** |
| **Description:** | Fires when a cy command is first invoked and enqueued to be run later. Useful for debugging purposes if you're confused about the order in which commands will execute. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | command:start |
| **Yields:** | command instance **(Object)** |
| **Description:** | Fires when cy begins actually running and executing your command. Useful for debugging and understanding how the command queue is async. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | command:end |
| **Yields:** | command instance **(Object)** |
| **Description:** | Fires when cy finishes running and executing your command. Useful for debugging and understanding how commands are handled. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | command:retry |
| **Yields:** | retry options **(Object)** |
| **Description:** | Fires whenever a command begins its [retrying routines](https://docs.cypress.io/guides/core-concepts/retry-ability). This is called on the trailing edge after Cypress has internally waited for the retry interval. Useful to understand **why** a command is retrying, and generally includes the actual error causing the retry to happen. When commands fail the final error is the one that actually bubbles up to fail the test. This event is essentially to debug why Cypress is failing. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | log:added |
| **Yields:** | log attributes **(Object)**, whether Cypress is in interactive mode (running via cypress open) **(Boolean)** |
| **Description:** | Fires whenever a command emits this event so it can be displayed in the Command Log. Useful to see how internal cypress commands utilize the [Cypress.log()](https://docs.cypress.io/api/cypress-api/cypress-log) API. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | log:changed |
| **Yields:** | log attributes **(Object)**, whether Cypress is in interactive mode (running via cypress open) **(Boolean)** |
| **Description:** | Fires whenever a command's attributes changes. This event is debounced to prevent it from firing too quickly and too often. Useful to see how internal cypress commands utilize the [Cypress.log()](https://docs.cypress.io/api/cypress-api/cypress-log) API. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | test:before:run |
| **Yields:** | test attributes **(Object)**, runnable instance **(Object)** |
| **Description:** | Fires before the test and all **before** and **beforeEach** hooks run. |

| **Event** | **Details** |
| --- | --- |
| **Name:** | test:after:run |
| **Yields:** | test attributes **(Object)**, runnable instance **(Object)** |
| **Description:** | Fires after the test and all **afterEach** and **after** hooks run. |

### Other Events

There are a myriad of other events Cypress fires to communicate with the Node server process, automation servers, mocha, the application, and the reporter. They are strictly internal to the way Cypress works and not useful for users.

## Binding to Events

Both the global Cypress and cy objects are standard Node event emitters. That means you can use the following methods to bind and unbind from events.

* [on](https://nodejs.org/api/events.html#events_emitter_on_eventname_listener)
* [once](https://nodejs.org/api/events.html#events_emitter_once_eventname_listener)
* [removeListener](https://nodejs.org/api/events.html#events_emitter_removelistener_eventname_listener)
* [removeAllListeners](https://nodejs.org/api/events.html#events_emitter_removealllisteners_eventname)

It's important to understand why you'd want to bind to either Cypress or cy.

### Cypress

Cypress is a global object that persists for the entirety of all of your tests. Any events you bind to Cypress will apply to all tests, and will not be unbound unless you manually unbind them.

This is useful when you're debugging and want to add a single "catch-all" event to track down things like test failures, or uncaught exceptions from your application.

### cy

The cy object is bound to each individual test. Events bound to cy will **automatically** be removed when the test ends. You don't have to worry about cleanup, and your event listeners will only be called for the duration of the single test.

## Examples

### Uncaught Exceptions

#### To turn off all uncaught exception handling

A great place to put this configuration is in the [supportFile](https://docs.cypress.io/guides/core-concepts/writing-and-organizing-tests#Support-file), since it is loaded before any test files are evaluated.

Cypress.on('uncaught:exception', (err, runnable) => {  
 // returning false here prevents Cypress from  
 // failing the test  
 *return* false  
})

### To conditionally turn off uncaught exception handling for a certain error

A great place to put this configuration is in the [supportFile](https://docs.cypress.io/guides/core-concepts/writing-and-organizing-tests#Support-file), since it is loaded before any test files are evaluated.

Cypress.on('uncaught:exception', (err, runnable) => {  
 // we expect a 3rd party library error with message 'list not defined'  
 // and don't want to fail the test so we return false  
 *if* (err.message.includes('list not defined')) {  
 *return* false  
 }  
 // we still want to ensure there are no other unexpected  
 // errors, so we let them fail the test  
})

### To conditionally turn off uncaught exception handling unhandled promise rejections

A great place to put this configuration is in the [supportFile](https://docs.cypress.io/guides/core-concepts/writing-and-organizing-tests#Support-file), since it is loaded before any test files are evaluated.

Cypress.on('uncaught:exception', (err, runnable, promise) => {  
 // when the exception originated from an unhandled promise  
 // rejection, the promise is provided as a third argument  
 // you can turn off failing the test in this case  
 *if* (promise) {  
 *return* false  
 }  
 // we still want to ensure there are no other unexpected  
 // errors, so we let them fail the test  
})

#### To catch a single uncaught exception

it('is doing something very important', (done) => {  
 // this event will automatically be unbound when this  
 // test ends because it's attached to 'cy'  
 cy.on('uncaught:exception', (err, runnable) => {  
 expect(err.message).to.include('something about the error')  
  
 // using mocha's async done callback to finish  
 // this test so we prove that an uncaught exception  
 // was thrown  
 done()  
  
 // return false to prevent the error from  
 // failing this test  
 *return* false  
 })  
  
 // assume this causes an error  
 cy.get('button').click()  
})

### Catching Test Failures

#### Debug the moment a test fails

If you want to debug when any test fails you'll likely want to put this in a [supportFile](https://docs.cypress.io/guides/core-concepts/writing-and-organizing-tests#Support-file), or at the top of an individual spec file.

Cypress.on('fail', (error, runnable) => {  
 *debugger*  
  
 // we now have access to the err instance  
 // and the mocha runnable this failed on  
  
 *throw* error // throw error to have test still fail  
})  
  
it('calls the "fail" callback when this test fails', () => {  
 // when this cy.get() fails the callback  
 // is invoked with the error  
 cy.get('element-that-does-not-exist')  
})

Read [Cypress Metaprogramming](https://glebbahmutov.com/blog/cy-metaprogramming/) for more examples.

### Page Navigation

### [End-to-End Only](https://docs.cypress.io/guides/core-concepts/testing-types" \l "What-is-E2E-Testing)

#### Test that your application was redirected

// app code  
$('button').on('click', (e) => {  
 // change the page programmatically  
 window.location.href = '/some/new/link'  
})  
  
// test code  
it('redirects to another page on click', (done) => {  
 // this event will automatically be unbound when this  
 // test ends because it's attached to 'cy'  
 cy.on('window:before:unload', (e) => {  
 // no return value on the event  
 expect(e.returnValue).to.be.*undefined*  
 })  
  
 cy.on('window:unload', (e) => {  
 // using mocha's async done callback to finish  
 // this test so we are guaranteed the application  
 // was unloaded while navigating to the new page  
 done()  
 })  
  
 // click the button causing the page redirect  
 cy.get('button').click()  
})

### Window Before Load

### [End-to-End Only](https://docs.cypress.io/guides/core-concepts/testing-types" \l "What-is-E2E-Testing)

#### Modify your Application before it loads after page transitions

it('can modify the window prior to page load on all pages', () => {  
 // create the stub here  
 *const* ga = cy.stub().*as*('ga')  
  
 // prevent google analytics from loading  
 // and replace it with a stub before every  
 // single page load including all new page  
 // navigations  
 cy.on('window:before:load', (win) => {  
 Object.defineProperty(win, 'ga', {  
 configurable: false,  
 get: () => ga, // always return the stub  
 set: () => {}, // don't allow actual google analytics to overwrite this property  
 })  
 })  
  
 cy  
 // window:before:load will be called here  
 .visit('/first/page')  
  
 .then((win) => {  
 // and here  
 win.location.href = '/second/page'  
 })  
  
 // and here  
 .get('a')  
 .click()  
})

### Window Confirm

#### Control whether you accept or reject confirmations

This enables you to test how your application reacts to accepted confirmations and rejected confirmations.

// app code  
$('button').on('click', (e) => {  
 *const* one = confirm('first confirm')  
  
 *if* (one) {  
 *const* two = confirm('second confirm')  
  
 *if* (!two) {  
 *const* three = confirm('third confirm')  
  
 confirm('third confirm was ' + three)  
 }  
 }  
})  
  
// test code  
it('can control application confirms', (done) => {  
 *let* count = 0  
  
 // make sure you bind to this \*\*before\*\* the  
 // confirm method is called in your application  
 //  
 // this event will automatically be unbound when this  
 // test ends because it's attached to 'cy'  
 cy.on('window:confirm', (str) => {  
 count += 1  
  
 *switch* (count) {  
 *case* 1:  
 expect(str).to.eq('first confirm')  
 // returning nothing here automatically  
 // accepts the confirmation  
 *case* 2:  
 expect(str).to.eq('second confirm')  
  
 // reject the confirmation  
 *return* false  
  
 *case* 3:  
 expect(str).to.eq('third confirm')  
  
 // don't have to return true but it works  
 // as well  
 *return* true  
  
 *case* 4:  
 expect(str).to.eq('third confirm was true')  
  
 // using mocha's async done callback to finish  
 // this test so we are guaranteed everything  
 // got to this point okay without throwing an error  
 done()  
 }  
 })  
  
 // click the button causing the confirm to fire  
 cy.get('button').click()  
})  
  
it('could also use a stub instead of imperative code', () => {  
 *const* stub = cy.stub()  
  
 // not necessary but showing for clarity  
 stub.onFirstCall().returns(*undefined*)  
 stub.onSecondCall().returns(false)  
 stub.onThirdCall().returns(true)  
  
 cy.on('window:confirm', stub)  
  
 cy.get('button')  
 .click()  
 .then(() => {  
 expect(stub.getCall(0)).to.be.calledWith('first confirm')  
 expect(stub.getCall(1)).to.be.calledWith('second confirm')  
 expect(stub.getCall(2)).to.be.calledWith('third confirm')  
 expect(stub.getCall(3)).to.be.calledWith('third confirm was true')  
 })  
})

### Window Alert

#### Assert on the alert text

Cypress automatically accepts alerts but you can still assert on the text content.

// app code  
$('button').on('click', (e) => {  
 alert('hi')  
 alert('there')  
 alert('friend')  
})  
  
it('can assert on the alert text content', () => {  
 *const* stub = cy.stub()  
  
 cy.on('window:alert', stub)  
  
 cy.get('button')  
 .click()  
 .then(() => {  
 expect(stub.getCall(0)).to.be.calledWith('hi')  
 expect(stub.getCall(1)).to.be.calledWith('there')  
 expect(stub.getCall(2)).to.be.calledWith('friend')  
 })  
})