Cypress App

wait

Wait for a number of milliseconds or wait for an aliased resource to resolve before moving on to the next command.

Syntax

```
cy.wait(time)
cy.wait(alias)
cy.wait(aliases)
cy.wait(time, options)
cy.wait(alias, options)
cy.wait(aliases, options)
```

```
// Specifying request and response types for aliased
intercepts
type UserReq = {}
type UserRes = {}
type ActivityReq = {}
type ActivityRes = {}

cy.intercept('/users/*').as('getUsers')
cy.intercept('/activities/*').as('getActivities')

// As templated types:
cy.wait<UserReq, UserRes>('@getUsers').then(({ request,
response }) => {
  request.body // will be of type UserReq
  response.body // will be of type UserRes
})

// As inferred types, with type `Interception` available
in `cypress/types/net-stubbing`
```

```
cy.wait('@getUsers').then(
  ({ request, response }: Interception<UserReq, UserRes>)
=> {
    request.body // will be of type UserReq
    response.body // will be of type UserRes
  }
)

// When passing an array of aliases, types must be
inferred:
cy.wait(['@getUsers', 'getActivities']).then(
  (
    interceptions: Array<
      Interception<UserReq | ActivityReq, UserRes |
ActivityRes>
    >
  ) => {
    interceptions.forEach(({ request, response }) => {
      request.body // will be of type UserReq |
ActivityReq
      response.body // will be of type UserRes |
ActivityRes
    })
  }
)
```

Usage

✓ Correct Usage

```
cy.wait(500)
cy.wait('@getProfile')
```

Arguments

> time *(Number)*

The amount of time to wait in milliseconds.

alias *(String)*

An aliased route as defined using the `.as()` command and referenced with the `@` character and the name of the alias.

Core Concept
You can read more about aliasing routes in our Core Concept Guide.

aliases *(Array)*

An array of aliased routes as defined using the `.as()` command and referenced with the `@` character and the name of the alias.

options *(Object)*

Pass in an options object to change the default behavior of `cy.wait()`.

Option	Default	Description
`log`	`true`	Displays the command in the **Command log**
`timeout`	`requestTimeout`, `responseTimeout`	Time to wait for `cy.wait()` to resolve before **timing out**
`requestTimeout`	`requestTimeout`	Overrides the global `requestTimeout` for this request. Defaults to `timeout`.
`responseTimeout`	`responseTimeout`	Overrides the global `responseTimeout` for this request. Defaults to `timeout`.

Yields ?

When given a `time` argument:

- `cy.wait()` yields the same subject it was given.
- It is unsafe to chain further commands that rely on the subject after `.wait()`.

When given an `alias` argument:

- `cy.wait()` 'yields an object containing the HTTP request and response properties of the request.

Examples

Time

Wait for an arbitrary period of milliseconds:

```
cy.wait(2000) // wait for 2 seconds
```

Anti-Pattern
You almost **never** need to wait for an arbitrary period of time. There are always better ways to express this in Cypress.

Read about best practices here.

Additionally, it is often much easier to use `cy.debug()` or `cy.pause()` when debugging your test code.

Alias

For a detailed explanation of aliasing, read more about waiting on routes here.

Wait for a specific request to respond

End-to-End Test **Component Test**

```
// Wait for the alias 'getAccount' to respond
// without changing or stubbing its response
cy.intercept('/accounts/*').as('getAccount')
cy.visit('/accounts/123')
cy.wait('@getAccount').then((interception) => {
  // we can now access the low level interception
  // that contains the request body,
  // response body, status, etc
})
```

Wait automatically increments responses

Each time we use `cy.wait()` for an alias, Cypress waits for the next nth matching request.

```
// stub an empty response to requests for books
cy.intercept('GET', '/books', []).as('getBooks')
cy.get('#search').type('Peter Pan')

// wait for the first response to finish
cy.wait('@getBooks')

// the results should be empty because we
// responded with an empty array first
cy.get('#book-results').should('be.empty')

// now the request (aliased again as `getBooks`) will
return one book
cy.intercept('GET', '/books', [{ name: 'Peter Pan'
}]).as('getBooks')

cy.get('#search').type('Peter Pan')

// when we wait for 'getBooks' again, Cypress will
// automatically know to wait for the 2nd response
cy.wait('@getBooks')
```

```
// we responded with one book the second time
cy.get('#book-results').should('have.length', 1)
```

Aliases

You can pass an array of aliases that will be waited on before resolving.

When passing an array of aliases to `cy.wait()`, Cypress will wait for all requests to complete within the given `requestTimeout` and `responseTimeout`.

End-to-End Test **Component Test**

```
cy.intercept('/users/*').as('getUsers')
cy.intercept('/activities/*').as('getActivities')
cy.intercept('/comments/*').as('getComments')
cy.visit('/dashboard')

cy.wait(['@getUsers', '@getActivities',
'@getComments']).then(
  (interceptions) => {
    // interceptions will now be an array of matching
requests
    // interceptions[0] <-- getUsers
    // interceptions[1] <-- getActivities
    // interceptions[2] <-- getComments
  }
)
```

Using `.spread()` to spread the array into multiple arguments.

```
cy.intercept('/users/*').as('getUsers')
cy.intercept('/activities/*').as('getActivities')
cy.intercept('/comments/*').as('getComments')
cy.wait(['@getUsers', '@getActivities',
'@getComments']).spread(
  (getUsers, getActivities, getComments) => {
    // each interception is now an individual argument
  }
)
```

Notes

Nesting

Cypress automatically waits for the network call to complete before proceeding to the next command.

```
// Anti-pattern: placing Cypress commands inside .then
callbacks
cy.wait('@alias')
  .then(() => {
    cy.get(...)
  })

// Recommended practice: write Cypress commands serially
cy.wait('@alias')
cy.get(...)

// Example: assert status from cy.intercept() before
proceeding
cy.wait('@alias').its('response.statusCode').should('eq',
200)
cy.get(...)
```

Read Guide: Introduction to Cypress

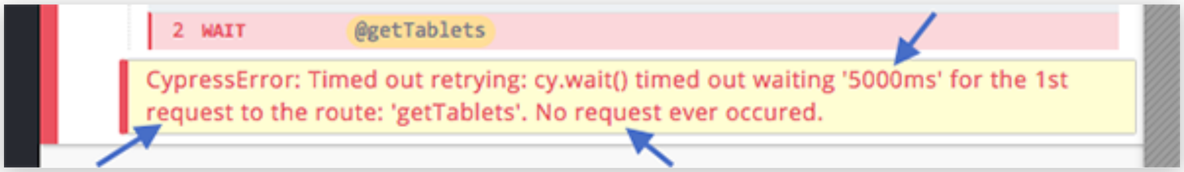
Timeouts

`requestTimeout` and `responseTimeout`

When used with an alias, `cy.wait()` goes through two separate "waiting" periods.

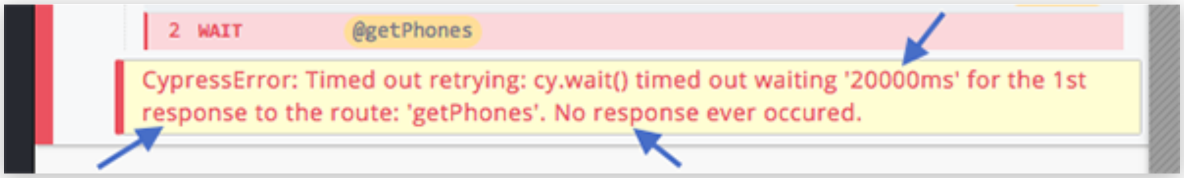
The first period waits for a matching request to leave the browser. This duration is configured by the `requestTimeout` option - which has a default of `5000` ms.

This means that when you begin waiting for an aliased request, Cypress will wait up to 5 seconds for a matching request to be created. If no matching request is found, you will get an error message that looks like this:


```
2 WAIT @getTablets
CypressError: Timed out retrying: cy.wait() timed out waiting '5000ms' for the 1st request to the route: 'getTablets'. No request ever occured.
```

Once Cypress detects that a matching request has begun its request, it then switches over to the 2nd waiting period. This duration is configured by the `responseTimeout` option - which has a default of `30000` ms.

This means Cypress will now wait up to 30 seconds for the external server to respond to this request. If no response is detected, you will get an error message that looks like this:


```
2 WAIT @getPhones
CypressError: Timed out retrying: cy.wait() timed out waiting '20000ms' for the 1st response to the route: 'getPhones'. No response ever occured.
```

This gives you the best of both worlds - a fast error feedback loop when requests never go out and a much longer duration for the actual external response.

Using an Array of Aliases

When passing an array of aliases to `cy.wait()`, Cypress will wait for all requests to complete within the given `requestTimeout` and `responseTimeout`.

Rules

Requirements ?

- When passed a `time` argument `cy.wait()` can be chained off of `cy` or off another command.
- When passed an `alias` argument `cy.wait()` requires being chained off of `cy`.

Assertions ?

- `cy.wait()` will only run assertions you have chained once, and will not retry.

Timeouts ?

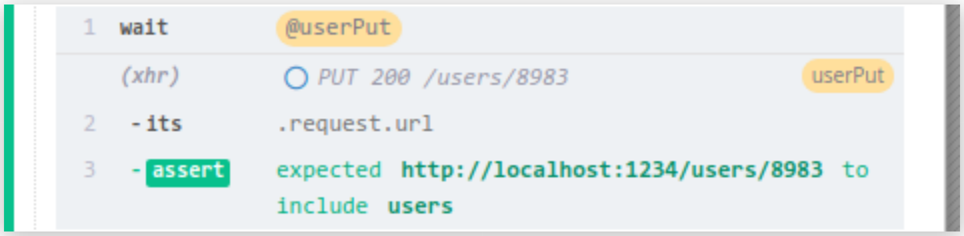
- `cy.wait()` can time out waiting for the request to go out.
- `cy.wait()` can time out waiting for the response to return.

Command Log

Wait for the PUT to users to resolve.

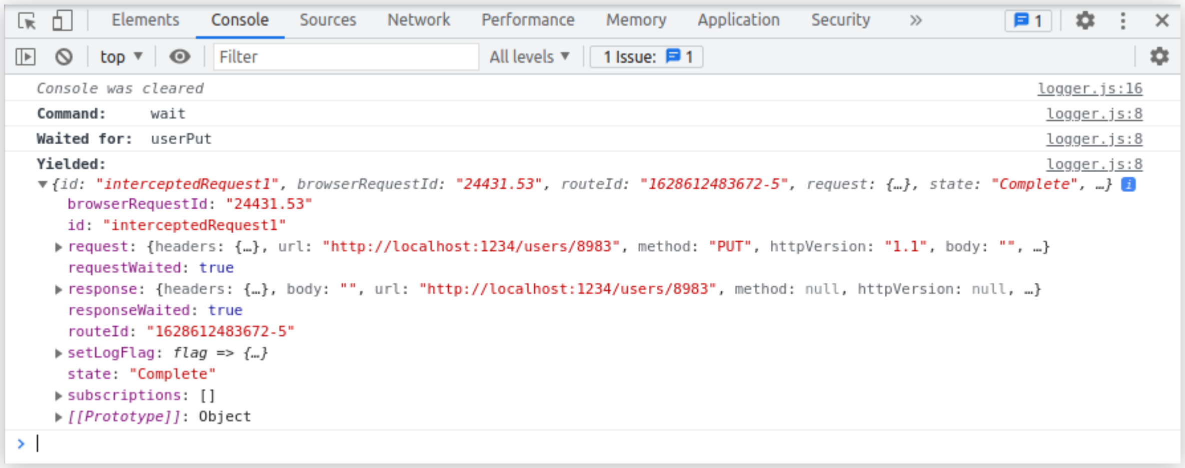
```
cy.intercept('PUT', /users/, {}).as('userPut')
cy.get('form').submit()
cy.wait('@userPut').its('request.url').should('include',
'users')
```

The `cy.wait()` will display in the Command Log as:



```
1  wait      @userPut
   (xhr)     ○ PUT 200 /users/8983                userPut
2  -its      .request.url
3  -assert   expected http://localhost:1234/users/8983 to
             include users
```

When clicking on `wait` within the command log, the console outputs the following:



```
Console was cleared                                                              logger.js:16
Command:     wait                                                                 logger.js:8
Waited for:  userPut                                                              logger.js:8
Yielded:                                                                          logger.js:8
▼{id: "interceptedRequest1", browserRequestId: "24431.53", routeId: "1628612483672-5", request: {…}, state: "Complete", …}
    browserRequestId: "24431.53"
    id: "interceptedRequest1"
  ▶ request: {headers: {…}, url: "http://localhost:1234/users/8983", method: "PUT", httpVersion: "1.1", body: "", …}
    requestWaited: true
  ▶ response: {headers: {…}, body: "", url: "http://localhost:1234/users/8983", method: null, httpVersion: null, …}
    responseWaited: true
    routeId: "1628612483672-5"
  ▶ setLogFlag: flag => {…}
    state: "Complete"
  ▶ subscriptions: []
  ▶ [[Prototype]]: Object
>
```

History

Version	Changes
3.1.3	Added `requestTimeout` and `responseTimeout` option
< 0.3.3	`cy.wait()` command added

See also

- `.as()`
- `cy.intercept()`
- `.spread()`

Edit this page

Last updated on ***Jan 14, 2025***

CONTENTS