What are Cypress Locators?

A Selector or Locator is an object that finds and returns web page items/elements on a page based on the query. In Automation, to perform any operation on the web elements first, you need to locate an element and perform an action on that element. The Locator or Selector helps to locate an element in the webpage. There are different types of locators, such as **id**, **CSS**, **XPath**, **tag-based selectors**, etc.

Cypress supports various locators such as tags, id, class, attributes, text, etc. Cypress also supports XPath Selectors; however, it needs installation of the third-party plugin **cypress-xpath**. Xpath has not been supported out of the box in the Cypress framework.

To fetch the HTML element in Cypress using different types of locators **cy.get()** method is used.

Table of Contents

- What are Cypress Locators?
- Get HTML Element by ID Selector in Cypress
- Get HTML element by Class in Cypress

- Get HTML element by Tag Name in Cypress
- Get HTML element by Attribute in Cypress
- Working with Multiple Elements in Cypress
- Useful Cypress Locator Functions
- Get the HTML Element by XPath in Cypress
- Executing Cypress Test Scripts using Browserstack

Get HTML Element by ID Selector in Cypress

ID is an attribute of an HTML tag, which is used to find an HTML element. Using the Cypress command – **cy.get()**, you can directly pass the id with prefix # and get the element.

Let's understand this using an example:

Consider code for an HTML element with **id** name **user_email_login** and **class** name **user_email_ajax**

```
<input id="user_email_login" class="user_email_ajax">
```

To get the HTML element by id in Cypress, use the following command:

```
cy.get('#user_email_login')
```

In this command, # is used as a prefix to id inside **cy.get()**

Once you are able to find the HTML element, you can perform operations on the elements such as **type**, **click**, etc., as seen in the example below:

```
cy.get('#user_email_login').type('myid98788');
```

In the above example, input **myid98788** is entered in the HTML element with ID **user_email_login**.

Get HTML element by Class in Cypress

Just like **ID**, **Class** is also an attribute of an HTML element, that is used as a locator or selector. Similar to how ID is directly passed with a prefix # using the Cypress command **cy.get()**, **class** can also be used to get the HTML element with **. (dot)** as a prefix inside **cy.get()**.

For example, consider code for an HTML element with **class** name **user_email_ajax**

```
<input id="user_email_login" class="user_email_ajax">
```

To get the HTML element by id in Cypress, use the following command:

```
cy.get('.user_email_ajax')
```

In this command, **. (dot)** is used as a prefix to class inside **cy.get()**

Get HTML element by Tag Name in Cypress

Using the Cypress command – **cy.get()**, you can directly get the element by passing the tag name.

Consider code for an HTML element with **tag** name **input**

```
<input id="user_email_login" class="user_email_ajax">
```

To get the HTML element by Tag name in Cypress, use the following command:

```
cy.get('input')
```

Get HTML element by Attribute in Cypress

Cypress provides a way to get the element by attribute name. Since it supports all the different types of CSS selectors, you can pass the CSS selectors inside the **cy.get()** command to get an element.

Let’s understand this using an example:

Consider code for an HTML element with **id** name **user_email_login** and **class** name **user_email_ajax**

```
<input id="user_email_login" class="user_email_ajax" type="text"
name="user_login">
```

In the above code, there is an attribute called **name.** To get an element using an attribute in Cypress, you can simply pass CSS selector to **cy.get()** as seen below:

```
cy.get('input[name="user_login"]')
```

However, id and class are also attributes. So, instead of using #, or **. (dot)** prefix, you can also get the element by using ID name or Class name by using ID and class as attributes.

- **Get element by ID using id as an attribute**

```
cy.get('input[id="user_email_login"]')
```

- **Get element by class name using class as an attribute**

```
cy.get('input[class="user_email"]')
```

The below example demonstrates Cypress Locators with actions:

```
describe('Cypress Locators Demo', () => {
it('Verify Error Message', () => {
cy.visit("https://www.browserstack.com/users/sign_in");
cy.get('#user_email_login').type('example@example.com')
cy.get('input[id="user_password"]').type('Mypassword123')
cy.get('#user_submit').click();
cy.get('.pass-input-placeholder > .input-wrapper > .error-msg > .msg-body > span').s
})
})
```

The above code navigates to the Browserstack Sign-in page and gets relevant locators. The following line gets an email textbox element with **ID** locator and allows the user to type the email address "**example@example.com**" in it

```
cy.get('#user_email_login').type('example@example.com')
```

Whereas, to get the password textbox element using the CSS attribute locators and type the password "**Mypassword123**", the following line is used.

```
cy.get('input[id="user_password"]').type('Mypassword123')
```

Then click on Submit button using the following command

```
cy.get('#user_submit').click();
```

The following is the Assertion for Error Message:

```
cy.get('.pass-input-placeholder > .input-wrapper > .error-msg > .msg-body > sp
```

Cypress CSS selectors with Regular Expression (RegEx) Patterns

Cypress also supports CSS regular expression pattern inside the **cy.get()** function

Symbol	Description	Example /Usage
*	Match all element that contains the value	**cy.get('[class*="react** Matches all class elem which contains the cla "react-class"
$	Match all element that ends with the value	**cy.get('[class$="react** Matches all class elem which ends with the cl "react-class"
^	Match all element that starts with the value	**cy.get('[class^="react** Matches all class elem which starts with the c "react-class"

Working with Multiple Elements in Cypress

When using the **cy.get()** function to fetch single or multiple elements, in case there are multiple matches, **cy.get()** returns all of them. Considering that you have to work with the first element, last element, or element at a specific index, then you need to use the **first()**, **last()**, or **eq()** functions along with **cy.get()**.

- **Get the first Element in Cypress**

The function **.first()** is used to get the first element in a set of elements

For Example:

```
cy.get('.msg-body').first()
```

In the above example, the command **cy.get('.msg-body')** might return more than one element, but by using **first()** it will always the first element within the set of the elements fetched.

- **Get the last Element in Cypress**

Similar to the above example, here **.last()** function is used to get the last element in a set of elements

For Example:

```
cy.get('.msg-body').last()
```

In the above example, the command **cy.get('.msg-body')** might return more than one element, but by using **last()** it will always be the last element within the set of the elements fetched.

- **Get the specific element by index in Cypress**

To get a specific element in the set of elements, using **eq()** function along with **cy.get('.msg-body')** as shown below

cy.get('.msg-body').eq(2)

- Get Element By Containing Text in Cypress

It might be difficult to find the element by text using CSS selectors, but Cypress provides a way to do that. You can simply use the **contains()** function along with the tag name and should be able to get the element.

The above approach is most useful to find the **href link** by Text in Cypress

```
For Example:
cy.get('a:contains("Forgot passoword")').first().click()
```

Note: The above example we are using contains() for <a> element, but you can use it for any element if you want to find the attribute or perform an action using text. There are other approaches discussed below.

- Get DOM Element containing Text in Cypress

Cypress provides **contains()** function, which can be used with **cy** command or can be chained with **cy.get()** command to get the DOM element as seen in the examples below

Example 1:

Find the element having Text Sign In

```
cy.contains('Sign In');
```

The above command searches for **Text 'Sign In'** inside the web page and returns the element. However, the command doesn't check which type of element it is.

Example 2:

Find the DOM element having Text Sign In

```
cy.get('a').contains('Sign In');
```

The above command searches for all **<a>** tags and returns the Element if **<a>** is having text '**Sign In**'. So, considering the first example, this approach is more specific to the tag name.

- **Using Regular expressions inside the cy.contains()**

Cypress allows the use of regular expressions inside **contains()** function. As seen in the below example, you can search for the element containing text that starts with b and has one or more word characters **(numbers, alphabets, and _)**

```
cy.contains(/^b\w+/)
```

To ignore the Case Sensitivity while using regular expressions inside **contains()** function, use **{ matchCase: false }**

```
cy.get('div').contains('capital sentence', { matchCase: false })
```

- **Get the descendent DOM elements of a specific selector**

To get the descended DOM element of a specific locator, use **cy.find()** function as seen in the example below:

For HTML Code

```
<ul id="parent">
<li class="first"></li>
<li class="second"></li>
</ul>
```

The following command returns all ordered list (**li**) elements inside the unordered list (**ul**) element

cy.get('#parent').find('li')

Useful Cypress Locator Functions

- **Get the Next Sibling in Cypress**

The **.next()** function returns the next sibling in Cypress as seen below

```
cy.get('#my_ele').next();
```

While, to get the next sibling which is an input tag, use the following command:

```
cy.get('#my_ele').next('input');
```

- **Get the Previous Sibling in Cypress**

The **.prev()** function returns the previous sibling in Cypress as seen below

```
cy.get('#my_ele').prev();
```

While, to get the previous sibling which is an input tag, use the following command:

```
cy.get('#my_ele').prev('input');
```

- **Get All Next Siblings in Cypress**

The **.nextAll()** function returns the next sibling in Cypress as seen below

```
cy.get('#my_ele').nextAll();
```

- **Get All Previous Sibling in Cypress**

The **.prevAll()** function returns the previous sibling in Cypress as seen below

```
cy.get('#my_ele').prevAll();
```

- **Get Any Siblings**

The **.siblings()** function is used to find the siblings in Cypress as seen below

```
cy.get('li').siblings('.active')
```

Or

```
cy.get('li').siblings()
```

- **Get The Siblings Until in Cypress**

The functions **.nextUntil()** and **.prevUntil()** allows you to get all the siblings before or after a certain condition. For example, to get all the next sibling elements until the element class name is **'my_class_name'**, use the command as seen below:

```
cy.get('li').nextUntil('.my_class_name')
```

Similarly, to get previous sibling element until the element class name is **'my_class_name'**, use the command as seen below:

```
cy.get('li').prevUntil('.my_class_name')
```

- **Within Command in Cypress**

The **.within()** function scopes all subsequent cy commands within this element as seen in the example below:

For HTML Code

```
<form>
<input name="email" type="email">
<input name="password" type="password">
<button type="submit">Login</button>
</form>
```

The following command will search for the elements only within the form, instead of searching the whole code, and returns only the relevant elements within the form,

```
cy.get('form').within(($form) => {
// cy.get() will only search for elements within form,
// not within the entire document
cy.get('input[name="email"]').type('eaxample@email.com')
cy.get('input[name="password"]').type('mypassword')
cy.root().submit()
})
```

Get the HTML Element by XPath in Cypress

As mentioned earlier Cypress doesn't support XPath out of the box, however, if you are already familiar with XPath then you can use the third-party plugin **cypress-xpath** to get the HTML element by XPath by using the following steps:

Step 1: Install the **cypress-xpath** plugin using the below command:

```
npm install -D cypress-xpath
```

Step 2: Set Up XPath plugin

Navigate to **cypress/support/index.js** and add the below line

```
require('cypress-xpath')
```

Step 3: Use **cy.xpath()** command in the script as seen in the example below

```
it('finds list items', () => {
cy.xpath('//ul[@class="todo-list"]//li')
.should('have.length', 3)
})
```

To get specific elements using XPath you can also chain the **cy.xpath()** to another command as seen below

```
it('finds list items', () => {
cy.xpath('//ul[@class="todo-list"]')
.xpath('./li')
.should('have.length', 3)
})
```

Executing Cypress Test Scripts using Browserstack

BrowserStack allows you to execute your Cypress test scripts or specs on multiple platforms and multiple browser versions. Interestingly you do not have to change Cypress specs, all you have to do is a simple configuration set up as shown in the following steps:

Step 1: Install BrowserStack Cypress plugin

Step 2: Create a **browserstack.json** file using the **browserstack-cypress init** command

Step 3: Enter the below code to the **browserstack.json** file

```
{
"auth": {
"username": "<my_username>",
"access_key": "<my_access_key>"
},
"browsers": [
{
"browser": "chrome",
"os": "Windows 10",
"versions": [
"latest",
"latest - 1"
]
}
],
"run_settings": {
"cypress_config_file": "./cypress.json",
"cypress_version": "9",
"project_name": "My sandbox project",
"build_name": "Build no. 1",
"parallels": "2",
}
}
```

Note: You can find the username and access key by logging into the BrowserStack website. You can also change the browser settings and platform settings from the **browserstack.json file**

Step 4: Configure the **cypress.json** file to include the **.js** files.

```
{
  "testFiles":["*.js"]
  }
```

Step 5: Execute your BrowserStack Test by using the below command

```
browserstack-cypress run –sync
```

Bear in mind that Cypress testing must be executed on real browsers for accurate results. Start running tests on 30+ versions of the latest browsers across Windows and macOS with BrowserStack. Use instant, hassle-free Cypress parallelization to run Cypress Parallel tests and get faster results without compromising on accuracy. Detect bugs before users do by testing software in real user conditions with BrowserStack.

Try Cypress Tests on BrowserStack

Automation Testing   Cypress

Was this post useful?   Yes, Thanks   Not Really

Related Articles

How to run Cypress Tests in Chrome and Edge

This article will demonstrate how to use Cypress browser website test cases on Google Chrome and Mic...

Learn More

How to Run Cypress Tests in Parallel

Learn parallel test execution on Cypress and to run Cypress group tests on real browsers on the Brow...

Learn More

Locators in Selenium: A Detailed Guide

Learn about different Locators in Selenium - ID, XPath, Name, DOM, Link, Tag & more that enables...

Learn More

Ready to try BrowserStack?

Over 6 million developers and 50,000 teams test on BrowserStack. Join them.

Contact Sales

PRODUCTS

- Live
- Automate
- Automate TurboScale
- Percy
- App Live
- App Automate
- App Percy
- Test Management
- Test Reporting & Analytics
- Accessibility Testing
- Accessibility Automation
- App Accessibility Testing
- Low Code Automation
- Bug Capture
- Requestly
- Website Scanner

WHY BROWSERSTACK

- Customers
- Case Studies
- Browsers & Devices
- Enterprise
- Data Centers
- Real Device Features
- Security

RESOURCES

- Support
- Status
- Release Notes
- Blog
- Events
- Community
- Meetups
- Champions
- Guide
- Partners
- Find a partner
- Trust Center
- Test University (Beta)

COMPANY

- About Us
- Careers
- Open Source
- Press
- Newsletter

SOCIAL

Contact Us

We are available 24 / 7

More Resources Cross Browser Testing • Selenium • Test Management • Emulators vs Real Device • Mobile App Testing

Test On Devices Test on iPhone • Test on iPad • Test on Galaxy • Test In IE • Test on Android • Test on iOS • Test on Right Devices • Mobile Emulators

Tools SpeedLab • Screenshots • Responsive • Nightwatch.js