Playwright Notes

A comprehensive reference guide for browser automation and testing with Playwright.

Browser Management

Starting Browsers

Chromium

```
const { chromium } = require('playwright');
const browser = await chromium.launch();
```

Chrome

```
const { chromium } = require('playwright');
const browser = await chromium.launch({ channel: 'chrome' });
```

Microsoft Edge

```
const { chromium } = require('playwright');
const browser = await chromium.launch({ channel: 'msedge' });
```

Firefox

```
const { firefox } = require('playwright');
const browser = await firefox.launch();
```

WebKit

```
const { webkit } = require('playwright');
const browser = await webkit.launch();
```

Closing Browser

```
await browser.close();
```

Context Management

Browser contexts are isolated browser sessions that don't share cookies, cache, or local storage.

Create Context

```
const context = await browser.newContext();
```

Close Context

```
await context.close();
```

Page Management

Opening Pages

```
const page = await context.newPage();
```

Handling New Pages/Tabs

```
// Set up listener before action
const newPagePromise = context.waitForEvent('page');

// Trigger action that opens new page
await page.locator('a[target="_blank"]').click();

// Wait for new page
const newPage = await newPagePromise;
```

Page Operations

```
// List all pages
const pages = context.pages();
```

```
// Bring page to front
await page.bringToFront();

// Close page
await page.close();

// Check if closed
const isClosed = page.isClosed();
```

Page Information

```
// Get URL
const url = page.url();

// Get title
const title = await page.title();
```

Page Assertions

```
// Requires: import { expect } from '@playwright/test'

// Assert URL equals value
await expect(page).toHaveURL('https://example.com/dashboard');

// Assert URL with function
await expect(page).toHaveURL(url => {
  return url.pathname === '/results'
    && url.searchParams.get('page') === '1';
});

// Assert title
await expect(page).toHaveTitle('Dashboard');
```

Navigation

```
// Navigate to URL
await page.goto('https://example.com');

// Reload page
await page.reload();

// Navigate back
await page.goBack();

// Navigate forward
await page.goForward();
```

Element Selection

Playwright offers multiple ways to select elements:

By CSS Selector

```
const element = page.locator('#element');
```

By Text Content

```
const element = page.getByText('Submit');
```

By Label

```
const element = page.getByLabel('Username');
```

By ARIA Role

```
const element = page.getByRole('button', { name: 'Submit' });
```

By Placeholder

```
const element = page.getByPlaceholder('Enter your email');
```

By Alt Text

```
const element = page.getByAltText('Company Logo');
```

By Title Attribute

```
const element = page.getByTitle('Close');
```

By Test ID

```
const element = page.getByTestId('submit-button');
```

Waiting for Elements

Wait States

```
// Wait for attached to DOM
await page.locator('#menu').waitFor({ state: 'attached' });

// Wait for visible
await page.locator('#menu').waitFor({ state: 'visible' });

// Wait with custom timeout
await page.locator('#menu').waitFor({
  state: 'visible',
  timeout: 30 * 60 * 1000 // 30 minutes
});

// Wait for hidden
await page.locator('#menu').waitFor({ state: 'hidden' });

// Wait for detached from DOM
await page.locator('#menu').waitFor({ state: 'detached' });
```

Element State & Assertions

Getting Element State

```
// Text content
const text = await page.locator('#element').textContent();

// Inner text
const text = await page.locator('#element').innerText();

// Inner HTML
const html = await page.locator('#element').innerHTML();
```

```
// Outer HTML
const html = await page.locator('#element').outerHTML();

// Attribute value
const href = await page.locator('#element').getAttribute('href');

// Input value
const value = await page.locator('#input').inputValue();

// Bounding box (x, y, width, height)
const box = await page.locator('#element').boundingBox();
```

Checking Element State

```
// Visibility
const isVisible = await page.locator('#element').isVisible();
const isHidden = await page.locator('#element').isHidden();

// Enabled/Disabled
const isEnabled = await page.locator('#element').isEnabled();
const isDisabled = await page.locator('#element').isDisabled();

// Checked (checkbox/radio)
const isChecked = await page.locator('#checkbox').isChecked();

// Editable
const isEditable = await page.locator('#input').isEditable();
```

Element Assertions

```
// Requires: import { expect } from '@playwright/test'

// Attached to DOM
await expect(page.locator('#element')).toBeAttached();

// Visibility
await expect(page.locator('#element')).toBeVisible();
await expect(page.locator('#element')).toBeHidden();

// Text content
await expect(page.locator('#element')).toContainText('Welcome Master Bruce');

// Case insensitive text
await expect(page.locator('#element')).toContainText(
  'wElComE mAster bRuCe',
  { ignoreCase: true }
);

// Negative assertion
await expect(page.locator('#element')).not.toContainText('Error');

// Input value
await expect(page.locator('#input')).toHaveValue('Hello World');

// Multi-select values
await expect(page.locator('#multi-select')).toHaveValues(['red', 'green']);

// CSS class
await expect(page.locator('#element')).toHaveClass("error");
await expect(page.locator('#element')).not.toHaveClass("error");

// CSS style
await expect(page.locator('#element')).toHaveCSS('display', 'block');

// Attributes
```

```
await expect(page.locator('#element')).toHaveAttribute('alt-text');
await expect(page.locator('#element')).toHaveAttribute(
  'alt-text', 'Company Logo'
);

// Checked state
await expect(page.locator('#checkbox')).toBeChecked();
await expect(page.locator('#checkbox')).not.toBeChecked();

// Enabled/Disabled
await expect(page.locator('#element')).toBeEnabled();
await expect(page.locator('#element')).toBeDisabled();

// Focus
await expect(page.locator('#element')).toBeFocused();
```

Interactions

Click Operations

```
// Standard click
await page.locator('#button').click();

// Right click
await page.locator('#button').click({ button: 'right' });

// Double click
await page.locator('#button').dblclick();

// Click with modifiers
await page.locator('#button').click({ modifiers: ['Control'] });

// Click at coordinates
await page.locator('#button').click({ position: { x: 10, y: 5 } });
```

Hover Operations

```
// Hover
await page.locator('#button').hover();

// Hover at coordinates
await page.locator('#button').hover({ position: { x: 10, y: 5 } });
```

Drag and Drop

```
// Simple drag and drop
await page.locator('#source').dragTo(page.locator('#target'));

// Drag and drop with offset
await page.locator('#source').dragTo(page.locator('#target'), {
  sourcePosition: { x: 10, y: 5 }, // relative to source top-left
  targetPosition: { x: 10, y: 5 }  // relative to target top-left
});
```

Mouse Operations

```
// Move mouse
await page.mouse.move(100, 200);

// Click at position
await page.mouse.click(100, 200);

// Right click
await page.mouse.click(100, 200, { button: 'right' });

// Double click
await page.mouse.dblclick(100, 200);

// Press and release
```

```
await page.mouse.down();
await page.mouse.up();

// With options
await page.mouse.down({ button: 'right' });
await page.mouse.up({ button: 'right' });

// Scroll wheel
await page.mouse.wheel(0, 100);
```

Other Element Interactions

```
// Focus
await page.locator('#input').focus();

// Blur
await page.locator('#input').blur();

// Scroll into view
await page.locator('#element').scrollIntoViewIfNeeded();

// Select text
await page.locator('#element').selectText();
```

Form Elements

Text Input

```
// Fill input
await page.locator('#input').fill('Hello World');

// Type with Enter
await page.locator('#input').press('Enter');

// Key chord
await page.locator('#input').press('Control+A');

// Type with delay
await page.locator('#input').pressSequentially(
  'Hello',
  { delay: 100 }
);

// Clear input
await page.locator('#input').clear();
```

Checkboxes

```
// Check
await page.locator('#checkbox').check();

// Uncheck
await page.locator('#checkbox').uncheck();
```

Dropdowns

```
// Select single option
await page.locator('.select-color').selectOption('Red');

// Select multiple options
await page.locator('.select-color').selectOption(['Red', 'Green']);
```

Keyboard Operations

```
// Press key
await page.keyboard.press('Enter');

// Key chord
await page.keyboard.press('Control+A');

// Type text
await page.keyboard.type('Hello World');

// Type with delay
await page.keyboard.type('Hello World', { delay: 100 });

// Hold key
await page.keyboard.down('Shift');

// Release key
await page.keyboard.up('Shift');
```

File Operations

File Upload

```
let el = page.locator('input[type="file"]');

// Single file
await el.setInputFiles('photos/mountain.png');

// Multiple files
await el.setInputFiles([
  'photos/mountain.png',
  'photos/river.png'
]);

// From memory
await el.setInputFiles([
  {
    name: 'file1.txt',
    mimeType: 'text/plain',
    buffer: Buffer.from('Hello World')
  }
]);

// Clear file input
```

```
await el.setInputFiles([]);
```

File Download

```
// Set up listener
const downloadEvent = await page.waitForEvent('download');

// Trigger download
await page.locator('#download-link').click();

// Save file
const download = await downloadEvent;
await download.saveAs('report.pdf');
```

Advanced Features

JavaScript Evaluation

```
// Evaluate string
const sum = await page.evaluate('1 + 2');

// Evaluate function
await page.evaluate(() => alert('Hello World'));

// With arguments
const sum = await page.evaluate(([a, b]) => {
  return a + b;
}, [1, 2]);
```

Dialog Handling

```
// Accept alert
page.on('dialog', async dialog => {
  await dialog.accept();
});

// Prompt with text
page.on('dialog', async dialog => {
  await dialog.accept('Hello World');
});

// Dismiss confirmation
page.on('dialog', async dialog => {
  await dialog.dismiss();
});

// Conditional handling
page.on('dialog', async dialog => {
  if(dialog.message() === 'Are you sure?') {
    await dialog.accept();
  } else {
    await dialog.dismiss();
  }
});
```

Cookie Management

```
// Get all cookies
const cookies = await context.cookies();

// Add cookies
await context.addCookies([{
  name: 'session-id',
  value: 'abc123',
  domain: 'example.com',
  path: '/',
  expires: 24 * 60 * 60, // 1 day
  httpOnly: true,
  secure: true,
  sameSite: 'Lax'
}]);
```

```
// Clear all cookies
await context.clearCookies();

// Clear specific cookie
await context.clearCookies({ name: 'session-id' });

// Clear domain cookies
await context.clearCookies({ domain: 'example.com' });
```

Viewport Management

```
// Get viewport size (returns { width, height })
const viewportSize = page.viewportSize();

// Set viewport size
await page.setViewportSize({ width: 1280, height: 720 });
```

Screenshots

```
// Element screenshot
const element = page.locator('#element');
await element.screenshot({ path: 'element.png' });

// Viewport screenshot
await page.screenshot({ path: 'viewport.png' });

// Full page screenshot
await page.screenshot({ path: 'fullpage.png', fullPage: true });
```

Best Practices

Use semantic selectors: Prefer *getByRole*, *getByLabel*, *getByText* over CSS selectors when possible

Auto-waiting: Playwright automatically waits for elements to be actionable

Isolation: Use browser contexts for test isolation

Assertions: Use Playwright's built-in assertions for better error messages

Screenshots: Take screenshots on test failures for debugging

Timeouts: Adjust timeouts only when necessary for slow operations

Common Patterns

Login Flow

```
await page.goto('https://example.com/login');
await page.getByLabel('Username').fill('user@example.com');
await page.getByLabel('Password').fill('password123');
await page.getByRole('button', { name: 'Login' }).click();
await expect(page).toHaveURL('https://example.com/dashboard');
```

Form Submission

```
await page.getByLabel('Email').fill('test@example.com');
await page.getByLabel('Message').fill('Hello World');
await page.getByRole('button', { name: 'Submit' }).click();
await expect(page.getByText('Success')).toBeVisible();
```

Table Interaction

```
const rows = page.locator('table tbody tr');
const count = await rows.count();

for (let i = 0; i < count; i++) {
  const row = rows.nth(i);
  const text = await row.textContent();
  console.log(text);
}
```