|  |
| --- |
| Installation |

1. Install Node.js v 18.20.6 (LTS)

<https://nodejs.org/en/download>

A screenshot of a computer

Description automatically generated

1. Choose .msi installer
2. Select v18.20.6 from list

Direct download: <https://nodejs.org/dist/v18.20.6/node-v18.20.6-x64.msi>

Launch installer and follow prompts. Choosing defaults will be acceptable for most installations.

1. Install Playwright

Open a windows command prompt and create a directory where playwright will be installed, this will be the location where tests will be created

|  |
| --- |
| cd \  mkdir c:\playwright  cd \playwright |

Install playwright using npm (installed when node.js is installed)

|  |
| --- |
| npm init playwright@latest |

At each prompt, answer with the highlighted value

|  |
| --- |
| Need to install the following packages:  create-playwright@1.17.135  Ok to proceed? (y) y  Getting started with writing end-to-end tests with Playwright:  Initializing project in '.'  √ Do you want to use TypeScript or JavaScript? · TypeScript  √ Where to put your end-to-end tests? · tests  √ Add a GitHub Actions workflow? (y/N) · false  √ Install Playwright browsers (can be done manually via 'npx playwright install')? (Y/n) · true |

Once the installation is complete, you should see a directory structure similar to the following:

|  |
| --- |
| Directory of C:\playwright  02/13/2025 08:37 AM <DIR> .  02/13/2025 08:37 AM 97 .gitignore  02/13/2025 08:37 AM <DIR> node\_modules  02/13/2025 08:37 AM 2,814 package-lock.json  02/13/2025 08:37 AM 256 package.json  02/13/2025 08:37 AM 2,178 playwright.config.ts  02/13/2025 08:37 AM <DIR> tests  02/13/2025 08:37 AM <DIR> tests-examples |

1. Install Visual Studio Code

<https://code.visualstudio.com/>

A screenshot of a computer

Description automatically generated

Download here

Direct download link

https://code.visualstudio.com/sha/download?build=stable&os=win32-x64-user

Launch installer and follow prompts. Choosing defaults will be acceptable for most installations.

1. Install Playwright plugin for VS Code

Launch VS Code, click extensions icon in left navigation pane, search for ‘playwright’ in the filter. You will want to install ‘Playwright Test for VSCode’

A screenshot of a computer

AI-generated content may be incorrect.

1. Open the previously created Playwright project with VS Code.

File -> Open Folder

A screenshot of a computer

Description automatically generated

Navigate to c:\playwright, click ‘Select Folder’

A screenshot of a computer

Description automatically generated

You should now see the files created by the playwright init tool in the File Explorer, as well as an example test case in the Test Explorer

|  |  |
| --- | --- |
| File Explorer View | Test Explorer View |
| A screenshot of a computer program  Description automatically generated |  |

|  |
| --- |
| Configuring Playwright |

Playwright has many configuration options available, the options below we will add are basic options to capture test run information and extend the default time limit of a single test.

1. From the File Explorer, open ‘playwright.config.ts’

**Browser Setup**

Playwright can test against multiple browsers, for now we will configure the tool to use google chrome.

1. Look for the projects section and remove the following browser settings

|  |
| --- |
| {  name: 'chromium',  use: { ...devices['Desktop Chrome'] },  },  {  name: 'firefox',  use: { ...devices['Desktop Firefox'] },  },  {  name: 'webkit',  use: { ...devices['Desktop Safari'] },  }, |

1. Add the following to the project section

|  |
| --- |
| {  name: 'GoogleChrome',  use: {  channel: 'chrome',  viewport: { width: 1280, height: 600 },  },  }, |

**Collect trace when retrying the failed test**

1. In the defineConfig section find the ‘use’ entry and change/add the following

|  |
| --- |
| use: {  /\* Collect trace when retrying the failed test. See https://playwright.dev/docs/trace-viewer \*/  trace: 'retain-on-failure',  launchOptions: {  slowMo: 500  },  screenshot: 'only-on-failure',  video: 'retain-on-failure'  }, |

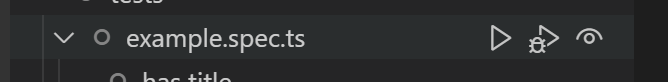
|  |
| --- |
| Running your First Test |

1. Navigate to the Test Explorer, expand tests. From here you are able to run and debug tests.

A screenshot of a computer program

Description automatically generated

1. Click on the ‘run’ icon to the right of ‘example.spec.ts’. This will launch a chrome browser and step through the individual tests (if there is more than one)



Once the test is complete, if successful you will see a few indicators.

1. This is an indicator of the test status
2. This section will show the number of pass / fail tests
3. This will show the detailed test status when a

A screenshot of a computer program

Description automatically generated

3

2

1

1. To view the report in HTML format, navigate to the project folder and then into ‘playwright-report’

A screenshot of a computer

AI-generated content may be incorrect.

Open index.html in a web browser.

A screenshot of a computer

AI-generated content may be incorrect.

|  |
| --- |
| Recording your First Test |

1. Click on the Test Explorer tab on the left navigation in VS Code
2. In the lower panel (Tools), select ‘Record New’

This will launch a new browser session and create a stub file in VS Code

1. Navigate to <https://example.com> in the browser launched by VS Code
2. Select the helper ‘Assert Text’ in the toolbar

A white box with black and black symbols

AI-generated content may be incorrect.

1. Click on the text ‘Example Domain’

Verify or change the expected text in the field and click the ‘check’

A screen shot of a computer

AI-generated content may be incorrect.

1. Click on ‘more information…’ link
2. Select the helper ‘Assert Text’ in the toolbar

A white box with black and black symbols

AI-generated content may be incorrect.

1. Click on text ‘Further Reading’

Verify or change the expected text in the field and click the ‘check’

A screenshot of a computer

AI-generated content may be incorrect.

1. Close the browser
2. We can now see a test-1.spec.ts appear in the VS Code explorer tab under the ‘tests’ subfolder. Opening this file will contain the following

|  |
| --- |
| import { test, expect } from '@playwright/test';  test('test', async ({ page }) => {  await page.goto('https://example.com/');  await page.getByRole('heading', { name: 'Example Domain' }).click();  await expect(page.getByRole('heading')).toContainText('Example Domain');  await page.getByRole('link', { name: 'More information...' }).click();  await expect(page.locator('h2')).toContainText('Further Reading');  }); |

1. In the VS Code File Explorer, right click on test-1.spec.ts and rename to ‘helloworld.spec.ts’

|  |
| --- |
| Recording a Simple CSI Test |

In this example, we will be recording the steps necessary to login to a CSI tenant and a specific site. We will then check the version number and fail the test if it does not match the expected value.

NOTE: The following instructions assume you are using the provided tenant at the SUN Elevate training session. Adjustments to url, username, password and site will be necessary to use the steps below with another tenant.

1. Click on the Test Explorer tab on the left navigation in VS Code
2. In the lower panel (Tools), select ‘Record New’

This will launch a new browser session and create a stub file in VS Code

1. Navigate to <https://mingle-portal.inforcloudsuite.com/v2/SLSGDENA209_AX1> in the browser launched by VS Code
2. Enter username and password provided to you for the tenant
3. Choose **SLSDENA209\_AX1\_DALS** for the site configuration, then click ok
4. Click the information icon in the navigation bar

A screenshot of a computer

AI-generated content may be incorrect.

1. Click ‘About’
2. Choose ‘Assert Text’ in the Playwright action bar

A close-up of a computer screen

AI-generated content may be incorrect.

1. Click the text box in the About modal and remove all text except for the version then click the checkmark

A screenshot of a computer

AI-generated content may be incorrect.

1. Unselect the ‘Assert Text’ in the playwright action bar. The action bar should now look like the following.

A screenshot of a computer

AI-generated content may be incorrect.

1. Click ‘Ok’ in the modal to close it
2. Click the ‘User Icon’ and then ‘Sign Out’

A screenshot of a computer

AI-generated content may be incorrect.

1. Click the ‘Record’ button in the playwright action bar to stop recording
2. Close the browser to complete recording of the test
3. In the VS Code File Explorer, right click on test-1.spec.ts and rename to ‘csi-version.spec.ts’

**Replaying Test**

1. In VS Code, switch to the Test Explorer
2. Click Run on the ‘csi-version.spec.ts’ test

A screenshot of a computer

AI-generated content may be incorrect.

1. You should see that this test will fail

A black background with white text

AI-generated content may be incorrect.

1. Open the ‘csi-version.spec.ts file in VS Code editor. If you followed the steps above, you should see a test case very similar to the following

|  |
| --- |
| import { test, expect } from '@playwright/test';  test('test', async ({ page }) => {  await page.goto('https://mingle-sso.inforcloudsuite.com/SLSGDENA209\_AX1/as/authorization.oauth2?client\_id=infor~5LfxcumSGSWnDkQVxIXdahNxt0J481cykg7FM-fiLAc\_OIDC&response\_type=code&redirect\_uri=https://mingle-portal.inforcloudsuite.com/sso/callback&scope=openid&state=SLSGDENA209\_AX1~cQ8tZ3VRfNFVE-WaexzgAx-8Pr8S\_Jpckw5fKcCvnhQ6AH1FmIUGa9UW4ccdBnoZ0JGkNWJ1cV0BzNK9X6Sea5MaksGx92DvvSpiSsUG9v8s-RfbZw\_RjCOg3J1OQJK8&code\_challenge=d4W4QqaUZWw0CUUxpfwnV25hhn4ZqPc5Hz213t4YqtY&code\_challenge\_method=S256');  await page.locator('body').click();  await page.getByRole('textbox', { name: 'Username' }).click();  await page.getByRole('textbox', { name: 'Username' }).fill('ElevateD1RF\_00');  await page.getByRole('textbox', { name: 'Password' }).click();  await page.getByRole('textbox', { name: 'Password' }).fill('SUNConf@dminD1RF');  await page.getByRole('button', { name: 'Sign in' }).click();  await page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().locator('#configCombo-trigger-picker').click();  await page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().getByRole('option', { name: 'SLSGDENA209\_AX1\_DALS' }).click();  await page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().getByRole('button', { name: 'Sign In' }).click();  await page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().locator('#mgtoolbarbutton-1096-btnEl').getByRole('button', { name: 'Help' }).click();  await page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().getByLabel('About', { exact: true }).getByRole('menuitem', { name: 'About' }).click();  await expect(page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().locator('#box-1202')).toContainText('2025.01.03.4');  await page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().getByRole('button', { name: 'OK' }).click();  await page.locator('#osp-nav-user-profile').click();  await page.getByRole('menuitem', { name: 'Sign out' }).click();  await page.goto('https://mingle-portal.inforcloudsuite.com/session/slo/signoutsuccess?tenantId=SLSGDENA209\_AX1');  }); |

The reason for the failure is Playwright did not record the original URL used to navigate to the tenant but instead captured a redirect

1. In the ‘csi-version.spec.ts’ file, replace the highlighted line in step 4 with the following

|  |
| --- |
| await page.goto('https://mingle-portal.inforcloudsuite.com/v2/SLSGDENA209\_AX1/5e29a9ce-6356-4b64-a59b-207c7fb1b5c7'); |

The GUID is important and is per tenant specific. The purpose of the GUID is to launch InforOS with a specific application as the default, in this case CSI. You can get the GUID for a specific tenant by logging into the tenant, launching the desired application then copying the URL from the browser.

1. Replay the test again, this time you should see that it passes

A black background with white text

AI-generated content may be incorrect.

1. Looking at the test script again, there is another area that could cause failures when running the script against different tenants. An example of this would be a script created against the TST tenant would not work as is with the TRN or PRD tenant. The highlighted text will change between tenants.

|  |
| --- |
| await page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().locator('#configCombo-trigger-picker').click(); |

If we inspect the HTML, we can see that CSI is actually embedded in an iframe and the name is generated by InforOS. It does not change per tenant but is different between tenants.

A screenshot of a computer

AI-generated content may be incorrect.

We will address this challenge in the tips section.

Recording a CSI Form Test

In this example, we will be recording the steps necessary to login to a CSI tenant and a specific site. We will then launch a form, retrieve data, make changes and save results.

NOTE: The following instructions assume you are using the provided tenant at the SUN Elevate training session. Adjustments to url, username, password and site will be necessary to use the steps below with another tenant.

1. Click on the Test Explorer tab on the left navigation in VS Code
2. In the lower panel (Tools), select ‘Record New’

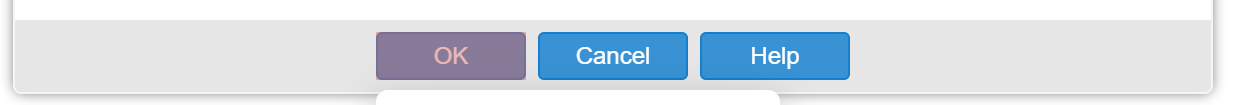
This will launch a new browser session and create a stub file in VS Code

1. Navigate to <https://mingle-portal.inforcloudsuite.com/v2/SLSGDENA209_AX1/5e29a9ce-6356-4b64-a59b-207c7fb1b5c7> in the browser launched by VS Code
2. Enter username and password provided to you for the tenant
3. Choose **SLSDENA209\_AX1\_DALS** for the site configuration, then click ok
4. Click the ‘Open Form’ button in the navigation toolbar
5. In the ‘Filter’ field, type ‘Customers’ and press ‘tab’
6. Click on ‘Customers.gen2’ in the Name column

A screenshot of a computer

AI-generated content may be incorrect.

1. Click Ok



1. Remove the Filter by clicking the ‘Filter in Place’ button in the navigation toolbar

A screenshot of a computer

AI-generated content may be incorrect.

1. In the customer search field, enter a valid customer number and press enter.

A screenshot of a phone

AI-generated content may be incorrect.

1. Click the Order Contact field and enter a new name, ensure you press tab after entering the name
2. Click the ‘Save’ Icon to save the record

A close-up of a logo

AI-generated content may be incorrect.

1. Close the Customers tab

A screen shot of a computer

AI-generated content may be incorrect.

1. Logout of the Tenant

A screenshot of a computer

AI-generated content may be incorrect.

1. Click the ‘Record’ button in the Playwright navigation bar to stop the recording

A screenshot of a computer

AI-generated content may be incorrect.

1. Close the browser to complete recording of the test
2. In the VS Code File Explorer, right click on test-1.spec.ts and rename to ‘csi-customer.spec.ts’

**Replaying Test**

1. In VS Code, switch to the Test Explorer
2. Open the ‘csi-customer.spec.ts’ test file
3. Ensure that the page.goto url is the following

|  |
| --- |
| await page.goto('https://mingle-portal.inforcloudsuite.com/v2/SLSGDENA209\_AX1/5e29a9ce-6356-4b64-a59b-207c7fb1b5c7'); |

1. Switch to the Test Explorer
2. Click Run on the ‘csi-version.spec.ts’ test

A black screen with white text

AI-generated content may be incorrect.

1. You should see that this test will fail. The reason this test failed can be found in the VS Code ‘Test Results’ windows at the bottom of the IDE. Alternatively, you can navigate to the project folder and under the ‘playwright-report’ folder, open index.html for details.

A screen shot of a computer

AI-generated content may be incorrect.

In this case, our test failed because the save button did not become active. The reason is playwright used the same name as was previously in the field so that no change to the form was detected.

There are 2 ways to solve this problem.

1. Detect if the save button is active and only simulate a save if available, otherwise throw an error. To make this change, look for the following line in the csi-customer.spec.ts file

|  |
| --- |
| await page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().getByLabel('Save', { exact: true }).getByRole('button', { name: 'Save' }).click(); |

And replace with the following

|  |
| --- |
| const isEnabled = await page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().getByLabel('Save', { exact: true }).getByRole('button', { name: 'Save' }).isEnabled();  if(isEnabled) {  await page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().getByLabel('Save', { exact: true }).getByRole('button', { name: 'Save' }).click();  } else {  throw new Error('Test Failed. Save button not active - see logs for details');  } |

The downside of this solution is that the test will never complete successfully unless another process first changes the name to a value different than what is in the test script.

1. Randomly generate names to enter into the form field.

Add the following to the top of the test script, just under the import statement

|  |
| --- |
| import { faker } from "@faker-js/faker";  export function randomName() {  const firstName = [  "Steve",  "Gary",  "John",  "Frank"  ]  const lastName = [  "Smith",  "Yellow",  "Pink",  "White",  ]  return firstName[faker.number.int({ min: 0, max: firstName.length - 1 })] + " " + lastName[faker.number.int({ min: 0, max: lastName.length - 1 })]  } |

This function, when called will return a random name combined from the first and last name arrays.

Look for the line that fills the Order Contact text box and replace the hard-coded name with the function name ‘randomName()’. The results should look similar to the following

|  |
| --- |
| await page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().getByRole('textbox', { name: 'Order Contact:' }).fill(randomName()); |

\*\* A new dependency was added and we need to ensure that it installed into the module library. To do so, open a command prompt, navigate to the project folder and execute the following command

|  |
| --- |
| npm install @faker-js/faker |

Using this method, a name will generally be unique from the current name populated in the form, so when the new name is entered, the form will notice a change and allow the test script to save the form and continue to success.

1. Make changes to the form using option ‘b’ from above
2. Run the ‘csi-customer.spec.ts’ test again, this time it should succeed

A screenshot of a computer

AI-generated content may be incorrect.

|  |
| --- |
| Tips and Tools |

In this section we will review tips to make scripts more maintainable and work for multiple target tenants.

**Hard-coded url, username, passwords**

When recording playwright scripts, all text entered is stored directly in the test case. In some cases, we do not want to do this as they could contain sensitive data or information that is only applicable to a specific tenant. Use the following process to store sensitive and or tenant specific information in an environment file.

First, we will need to install the dotenv package by going into the project folder with windows command prompt.

|  |
| --- |
| cd \playwright  npm install dotenv @playwright/test |

You will notice a new dependency in the packages.json after installing this package.

A screen shot of a computer program

AI-generated content may be incorrect.

After installing dotenv package we need to load it. The below function uses dotenv to load environment variables from a .env file. Finally, it exports the globalSetup function as the default export of the module. Create a file called globalSetup.ts in the helpers folder of the project.

|  |
| --- |
| //globalSetup.ts const { FullConfig } = require("@playwright/test"); const dotenv = require('dotenv');  async function globalSetup(config) {  dotenv.config({  path: '.env',  override: true  });  }   module.exports = globalSetup; |

Finally, to use the globalSetup, add the following to the defineConfig section of the playwright.config.ts file

|  |
| --- |
| globalSetup: 'helpers/globalSetup.ts', |

Create an .env file in the root of the project folder. Here you can define your environment specific data

|  |
| --- |
| TENANT = your\_env WEB\_URL = your\_URL  WEB\_URL\_GUID = your\_guid\_specific\_to\_tenant\_application USERNAME = your\_username PASSWORD = your\_password  SITE\_CONFIG = your\_site\_config  TARGET\_VERSION = CSI\_version\_expected |

Use process.env in tests to reference the variables defined in the .env file. For example, change the following in ‘csi-version.spec.ts’

|  |
| --- |
| await page.goto('https://mingle-portal.inforcloudsuite.com/v2/SLSGDENA209\_AX1/5e29a9ce-6356-4b64-a59b-207c7fb1b5c7');  await page.locator('body').click();  await page.getByRole('textbox', { name: 'Username' }).click();  await page.getByRole('textbox', { name: 'Username' }).fill('xxxxxxxxxxx');  await page.getByRole('textbox', { name: 'Password' }).click();  await page.getByRole('textbox', { name: 'Password' }).fill('xxxxxxxxxx');  await page.getByRole('button', { name: 'Sign in' }).click(); |

to the following will allow your test script to use the entries defined in .env. Note the special handling of variables in the username and password field. This allows you to pass a variable without the IDE complaining about an undefined value. Syntax is `${}` where ` is a backtick (grave accent).

|  |
| --- |
| await page.goto('https://' + process.env.WEB\_URL + '/v2/' + process.env.TENANT + '/' + process.env.WEB\_URL\_GUID);    await page.locator('body').click();    await page.getByRole('textbox', { name: 'Username' }).click();    await page.getByRole('textbox', { name: 'Username' }).fill(`${process.env.USERNAME}`);    await page.getByRole('textbox', { name: 'Password' }).click();    await page.getByRole('textbox', { name: 'Password' }).fill(`${process.env.PASSWORD}`);    await page.getByRole('button', { name: 'Sign in' }).click(); |

**Reuse previously recorded scripts in new scripts**

Currently, when recording a script, you are logging into the tenant each time and that information is stored in the script as steps. In this example, we will relocate the InforOS login process to a helper file and invoke from a previously created test script, ‘csi-version.spec.ts’

First, create a new file named appHelper.ts in the helpers folder. Edit the file and enter the following

|  |
| --- |
| import  { expect, Frame } from '@playwright/test';  export async function operatorSigin(page: any): Promise<void> {  } |

Copy the following from the csi-version.spec.ts and place into the new function

|  |
| --- |
| await page.goto('https://' + process.env.WEB\_URL + '/v2/' + process.env.TENANT + '/' + process.env.WEB\_URL\_GUID);    await page.locator('body').click();    await page.getByRole('textbox', { name: 'Username' }).click();    await page.getByRole('textbox', { name: 'Username' }).fill(`${process.env.USERNAME}`);    await page.getByRole('textbox', { name: 'Password' }).click();    await page.getByRole('textbox', { name: 'Password' }).fill(`${process.env.PASSWORD}`);    await page.getByRole('button', { name: 'Sign in' }).click(); |

The file should now look like this

|  |
| --- |
| import  { expect, Frame } from '@playwright/test';  export async function operatorSignin(page: any): Promise<void> {    await page.goto('https://' + process.env.WEB\_URL + '/v2/' + process.env.TENANT + '/' + process.env.WEB\_URL\_GUID);    await page.locator('body').click();    await page.getByRole('textbox', { name: 'Username' }).click();    await page.getByRole('textbox', { name: 'Username' }).fill(`${process.env.USERNAME}`);    await page.getByRole('textbox', { name: 'Password' }).click();    await page.getByRole('textbox', { name: 'Password' }).fill(`${process.env.PASSWORD}`);    await page.getByRole('button', { name: 'Sign in' }).click();  } |

Finally, modify the csi-version.spec.ts by adding a reference to appHelper.ts and replacing the copied text above with a reference to the operatorSignin function

|  |
| --- |
| import { operatorSignin } from '../helpers/appHelper.ts';  test('test', async ({ page }) => {  // login from helpers/appHelper.ts  await operatorSignin(page);  … |

**Auto-detect application iframe**

CSI is rendered in an iframe within the InforOS platform. The iframe is partly based on the application GUID and it will change between tenants. Use the following method below to allow playwright to automatically determine the iframe guid and replace with static references.

Create a new folder at the top level of your project called ‘helpers’, assuming the top level followed this tutorial, it would be c:\playwright\helpers. Inside the helpers folder, create a file called ‘frameHelper.ts’.

A screenshot of a computer

AI-generated content may be incorrect.

In frameHelper.ts, add the following.

|  |
| --- |
| import { Frame, Page } from '@playwright/test';  export async function getFrameLocator(page: Page, value?):Promise<Frame> {  if(value == undefined) {  value = "syteline"  }  await page.waitForTimeout(5000)  const frames = await page.frames();  let frame:any = null  for (let j = 0; j < frames.length; j++) {  const frameName = frames[j].name();  if (frameName.includes(value)) {  frame = frames[j]  }  }  if ( frame != null) {  return frame.name()  } else {  throw new Error("Frame not found with starting with " + value);  }  } |

Open a previously created test, in this example we will use ‘csi-version.spec.ts’

Add the following to the top of the file under the import section

|  |
| --- |
| import { getFrameLocator } from '../helpers/frameHelper.ts'; |

Directly under the sign-in action, shown here,

|  |
| --- |
| await page.getByRole('button', { name: 'Sign in' }).click(); |

add the following. This will ensure the site is loaded with the necessary iframes to load SyteLine and context widgets. The last 2 lines will find the iframe used to render syteline and create the text string used by playwright

|  |
| --- |
| // wait for nav bar to render so that we are sure iframes have been built  await expect(page.getByRole('button', { name: 'Application Launcher'  })).toBeVisible();  // retrieve syteline iframe name  const frameName = await getFrameLocator(page);  const iFrameVal = 'iframe[name="' + frameName + '"]'; |

You can now replace all occurrences of the static text that playwright used to build the locator for SyteLine iframe.

For example, if your page.locator looked like the following

|  |
| --- |
| await page.locator('iframe[name="syteline\_44\_5e29a9ce-6356-4b64-a59b-207c7fb1b5c7"]').contentFrame().locator('#configCombo-trigger-picker').click(); |

You will change to this

|  |
| --- |
| await page.locator(iFrameVal).contentFrame().locator('#configCombo-trigger-picker').click(); |