# Queries For Toli

Contents

[Queries For Toli 1](#_Toc526433026)

[Folder = POM-> Gas 1](#_Toc526433027)

[File = AccountGasPage 1](#_Toc526433028)

[Folder = POM-> Gas 2](#_Toc526433029)

[File = MultiInvoiceDownloadPage 2](#_Toc526433030)

[Folder = POM-> Gas 4](#_Toc526433031)

[File = AccountGasPage 4](#_Toc526433032)

[Folder = TestData 4](#_Toc526433033)

[File = TestDataRepository 4](#_Toc526433034)

[Folder = POM-> Gas ??? 5](#_Toc526433035)

[File = AcountSteps 5](#_Toc526433036)

[Suggestion(s) For Improvement 6](#_Toc526433037)

## Folder = POM-> Gas

### File = AccountGasPage

{

AccountSelection = By.Id("select2-selectAccount-container");

SiteSelection = By.Id("select2-selectSite-container");

AccountSelectionText = "Search by name or number";

SiteSelectionText = "Select a site";

}

public sealed override string SiteSelectionText { get; set; }

public sealed override string AccountSelectionText { get; set; }

public sealed override By SiteSelection { get; set; }

public sealed override By AccountSelection { get; set; }

What is AccountSelection and how does it work?

Does it somehow replace the .XPath, ID, etc in Driver.FindElement(By.?)

## Folder = POM-> Gas

### File = MultiInvoiceDownloadPage

public void EnterDate(string month, int year)

{

WaitFor(By.Id("MonthlyInvoiceDownLoadDate"));

IWebElement monthlyInvoiceDownloadDate = Driver.FindElement(By.Id("MonthlyInvoiceDownLoadDate"));

monthlyInvoiceDownloadDate.Click();

var monthSelection = $"{month} - {year}";

monthlyInvoiceDownloadDate.SendKeys(monthSelection);

}

WaitFor(By.Id("MonthlyInvoiceDownLoadDate"));

It seems from the BasePom file that this should return a collection of WebElements, but the returned value is not stored. How does that work?

var monthSelection = $"{month} - {year}";

This seems to concatenate the parameters into string, but how exactly is it working, what does the $ do?

It looks like these methods both do the same thing, although only one seems to be called. Is one of these redundant?

public void AssertInvoicesDoesNotExistOnDisk(string month, int year, string account)

{

var downloadsFolder = GetDownloadFolderPath();

var fileName = $"{account}\_{month} - {year}.zip";

Assert.IsFalse(FileExists(Path.Combine(downloadsFolder, fileName)), $"Invoices for account {account} for {month}, {year} found! Expected to have not downloaded this invoice");

}

public void InvoicesDoesNotExistOnDisk(string month, int year, string account)

{

var downloadsFolder = GetDownloadFolderPath();

var fileName = $"{account}\_{month} - {year}.zip";

Assert.IsFalse(FileExists(Path.Combine(downloadsFolder, fileName)), $"Invoices for account {account} for {month}, {year} found!");

}

var accountNumElement = WaitFor(By.Id("activeAccountNum")).FirstOrDefault();

FirsOrDefault is new to me.

How is the default value defined?

NOTE:

For your WebDriverWaits you are (quite correctly) using OpenQA.Selenium.Support.UI

However, this is being deprecated (no idea why!), seems that in future need to using DotNetSeleniumExtras (also need this for pageObjects)

public void ClickBackToSite()

{

if (!DoesElementExist(By.Id("BackToSiteLink"))) return;

Click(By.Id("BackToSiteLink"));

}

I assume that if the element does not exist then it will do nothing (return) and if it does then it will click it?

I’ve only seen return used to return a value, so a little confused by this use of it.

Same as this?

public void ClickBackToSite()

{

if (DoesElementExist(By.Id("BackToSiteLink");

Click(By.Id("BackToSiteLink"));

}

public void ClickBackToAccount()

{

WaitFor(By.Id("BackToAccountLink"));

var cickBackToAccount = Driver.FindElement(By.Id("BackToAccountLink"));

var js = Driver as IJavaScriptExecutor;

js?.ExecuteScript("arguments[0].click();", cickBackToAccount);

}

This will find the element and click on it, but why the use of java script executor and how exactly does that work?

## Folder = POM-> Gas

### File = AccountGasPage

public override void SearchAccount(string account)

{

SearchForAccount(account, "selectAccount");

}

F12

protected void SearchForAccount(string account, string selectedItem)

{

if (AccountSelected(account) || !AccountsArePopulated()) return;

ClickSelection(selectedItem);

EnterSearch(account);

}

F12

private void ClickSelection(string selectedItem)

{

Click(By.XPath($"//\*[@id='select2-{selectedItem}-container']"));

}

This looks like it is using the “selectAccount” string passed in at the top to build the Xpath to find the search box. Why not just use a normal find/wait?

Also, can you explain this?

if (AccountSelected(account) || !AccountsArePopulated()) return;

Also, are the later paramenters (we only pass the first 2 in here) for overrides?

protected void KeyPress(By by, string value, bool clearExisting = false, int? wait = null)

## Folder = TestData

### File = TestDataRepository

I can generally follow this, but it would be nice to understand the detail better (may need more than one session to understand it all)

Also, how can I find out which data file is being used?

public TestDataRepository()

{

string dataDirectory;

if (AppDomain.CurrentDomain.BaseDirectory.EndsWith(@"\Debug") || AppDomain.CurrentDomain.BaseDirectory.EndsWith(@"\Release"))

{

dataDirectory = $"{AppDomain.CurrentDomain.BaseDirectory}";

}

else

{

var codeBase = Assembly.GetExecutingAssembly().CodeBase;

var uri = new UriBuilder(codeBase);

var path = Uri.UnescapeDataString(uri.Path);

dataDirectory = Path.GetDirectoryName(path);

}

if (dataDirectory == null)

{

throw new ArgumentNullException(@"dataDirectory", new Exception($"Test data directory not found!"));

}

\_dataFolder = Path.Combine(dataDirectory, $"TestData\\{RunSettings.EnvironmentKey}\\");

}

public User GetUser(string userKey)

{

var users = \_xlsxService.ReadFromFile<User>($"{\_dataFolder}Users.xlsx");

var selectedUser = users.FirstOrDefault(u => u.UserKey == userKey);

if (selectedUser == null)

{

throw new ArgumentNullException(@"User", new Exception($"User {userKey} not found!"));

}

return \_xlsxService.ReadFromFile<User>($"{\_dataFolder}Users.xlsx").First(u => u.UserKey == userKey);

}

## Folder = POM-> Gas ???

### File = AcountSteps

[When(@"Accounts are populated")]

public void ThenSelectAccountDropDownExists()

{

\_accountGasPage.AccountsArePopulated();

}

F12

public virtual bool AccountsArePopulated()

{

Func<IWebElement, bool> condition = element => element.Text == AccountSelectionText;

var webElement = WaitForWithCondition(AccountSelection, condition, 30);

return webElement != null && webElement.Displayed;

}

Where do the values to populate AccountSelection and AccountSelectionText come from?

## Folder = Features->Power->Meter~Read

### File = MeterReadPower

Scenario:a #Submit Meter read for 1 Mpan -Single Meter Single Register

Given I am logged in as "*SmokeAdminSASSPower*"

And I have opened a site for electricity Account

When I click select for Electricity Account

Scenario:a #Submit Meter read for 1 Mpan -Single Meter Single Register For AdminUser

Given I am on Power account dashboard for group account *10000007* and account "*PN005160*" logged in as "*AdminUser*"

These 2 scenario snippets appear to do much the same thing, but in different ways.

One will use the given user and work out what group account and account/site ref to use from the spreadsheet. The other uses the values supplied.

Why have 2 different approaches?

## Suggestion(s) For Improvement

Instead of code like this

Click(By.XPath("/html/body/div[2]/div[2]/div/div/table/tbody/tr/td[7]/button"));

Where you cannot tell from looking at it what element is being clicked on

Have code like this:

[FindsBy (How = How.Xpath, Using = “/html/body/div[2]/div[2]/div/div/table/tbody/tr/td[7]/button”)]

IWebElment OKButton;

OKButton.Click();

(I think this needs PageFactory to work, but can also use the Driver.Find() approach to get the element, point is to include the element name when you interact with it)

Can logging be improved so it will say which class and method were being called when the error was encountered? (and any other information we think would be useful)