

HitachiQA Framework

The purpose of this document is to explain the various capabilities of the framework.

Table of Contents

HitachiQA Framework.....	1
Table of Contents	1
Setup	2
Clone repository (Command Line)	2
Authenticate (Azure Services).....	2
Chrome Webdriver (optional).....	3
Startup (BeforeTest, BeforeFeature, BeforeScenario).....	4
BeforeTest (Load Environment Files).....	4
BeforeTest (Azure Keyvault)	5
BeforeFeature (Invoking webdriver).....	6
Teardown (AfterTest, AfterFeature, AfterScenario)	7
AfterScenario (Gathering test results)	7
AfterFeature (Devops authentication).....	7
AfterFeature (Devops result submission)	8
AfterTest (disposal)	8
RestAPI	9
Authentication (from Webdriver)	9
Authentication (Bearer generation).....	9
SQL	10
Authentication:	10
Cosmos.....	11
Authentication:	11

Setup

Clone repository (Command Line)

1. Download/install Git

Download from: [Git - Downloads \(git-scm.com\)](https://git-scm.com)

Install in your system using installer's default selections.

Make sure to restart your system.

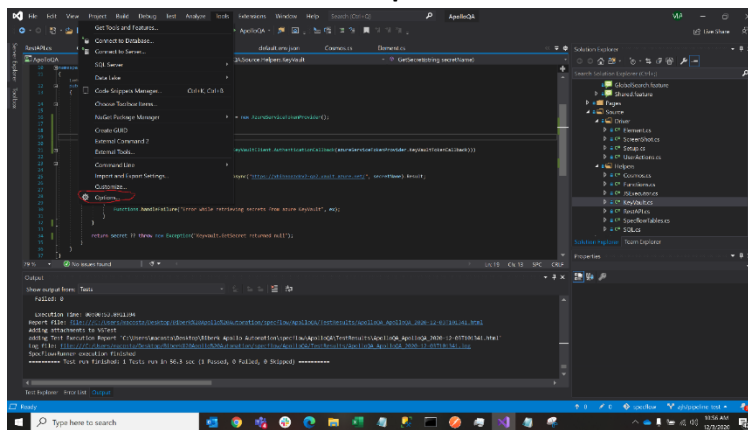
2. Clone source repository

Navigate to the desired directory

Run "git clone <https://github.com/Hitachi-QA/Hitachi-QA.git>"

Authenticate (Azure Services)

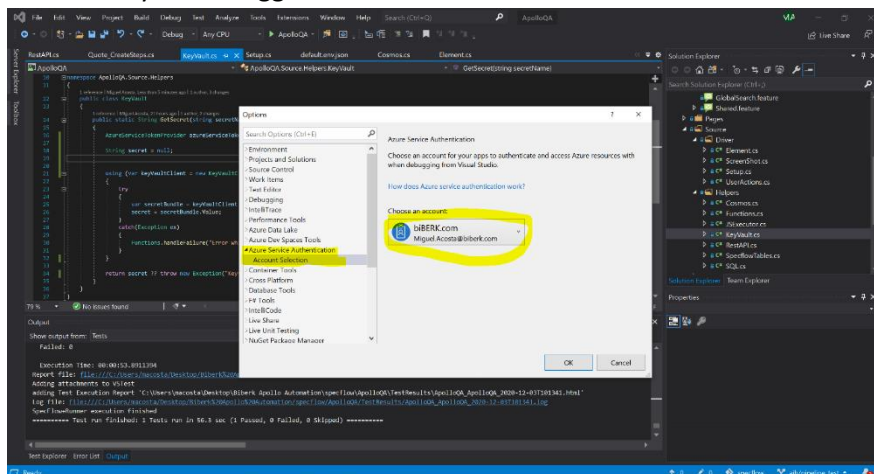
1. In Visual Studio Go to Tools-> Options:



2. Click on Azure Service Authentication

Navigate to the desired directory

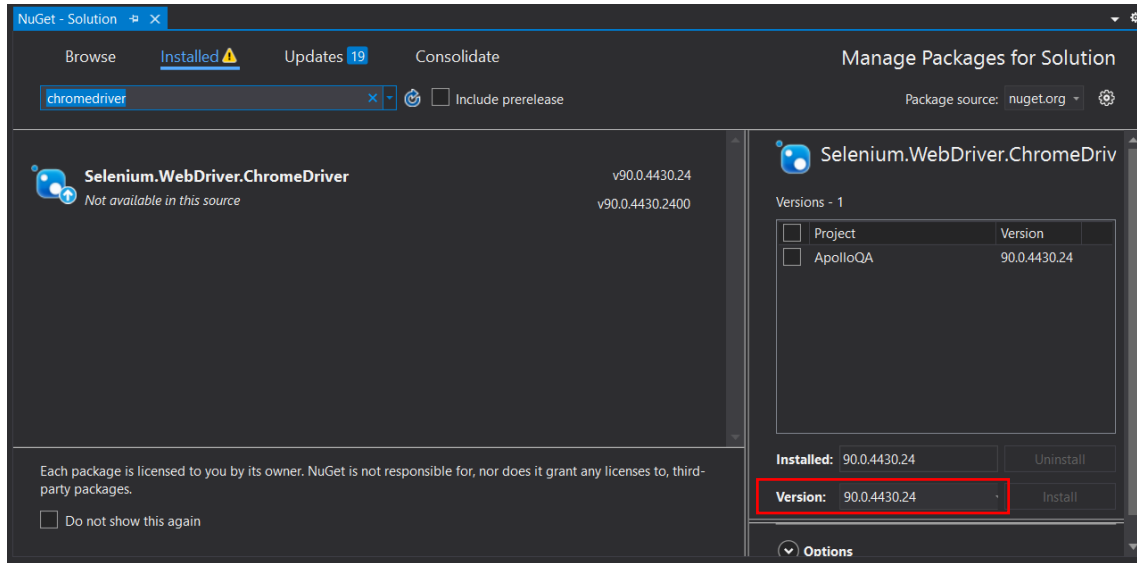
Make sure you are logged in with the account with access to the desired Keyvault.



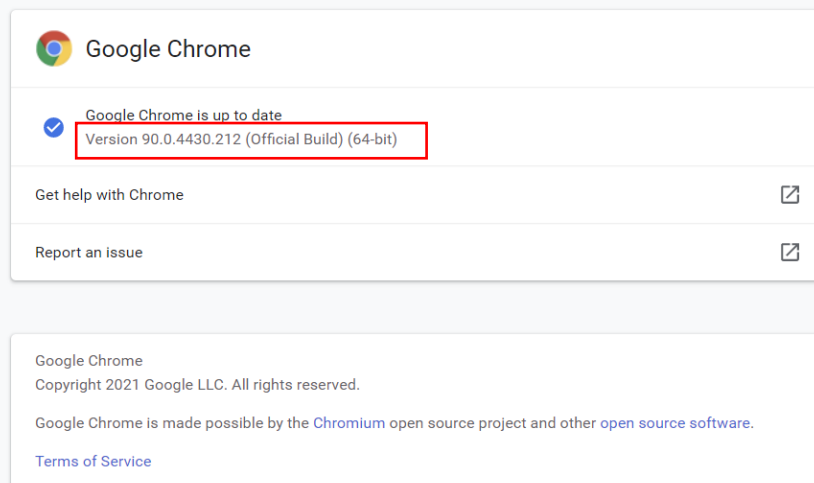
Chrome Webdriver (optional)

Steps required if the solution would need to use Chrome WebDriver

1. **Make sure that the Webdriver version is consistent between the solution and the Chrome web browser**



About Chrome



Startup (BeforeTest, BeforeFeature, BeforeScenario)

A set of static functions that will run as part of “startup”.

Order of execution is BeforeTest, then BeforeFeature, then BeforeScenario:

- BeforeTest: Runs once per execution prior to BeforeFeature.
- BeforeFeature: Runs once per Feature (file) prior to BeforeScenario.
- BeforeScenario: Runs prior to every scenario executed.

These are all located in “Source.Driver.Setup”.

BeforeTest (Load Environment Files)

1. **Environment File:**

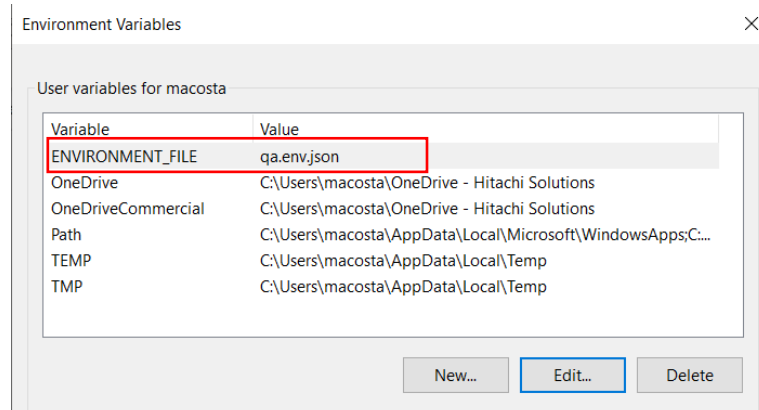
By Default, the system looks for “./default.env.json” in the solution.

2. **(optional) specify environment file to load**

Important: must be a single dimension json object <string, string>

the looks for variable “ENVIRONMENT_FILE” in the System & User level environment variables

E.g., the following will load variables from “./qa.env.json” in the solution.



BeforeTest (Azure Keyvault)

1. Keyvault connection:

If a variable was not found in the current environment variables, then the solution will attempt to load from Azure Keyvault using the following URI's to connect

First try: `KEYVAULT_URI` (Key vault dedicated to this solution, specific to the project)

Second Try: `APP_KEYVAULT_URI` (Application under test's Keyvault)

2. Keyvault Authentication:

Required: Authenticate (Azure Services)

The system uses Azure Service Authentication to generate a token in order to connect to the Keyvault

```
AzureServiceTokenProvider azureServiceTokenProvider = new AzureServiceTokenProvider();  
using (var keyVaultClient = new KeyVaultClient(new KeyVaultClient.AuthenticationCallback(azureServiceTokenProvider.KeyVaultTokenCallback)))
```

3. Keyvault Secrets:

any variable ending with `SECRETNAME` is interpreted to be holding the secret name to be loaded

Resulting in the variable `"SQLSTR"` to be loaded along with its value as an environment variable.

E.g.,

```
"SQL_CONNECTIONSTRING_SECRETNAME": "SQLSTR",
```

Will attempt to load `"SQLSTR"` from the `"KEYVAULT_URI"` if not successful, then attempt from

`"APP_KEYVAULT_URI"`.

Result: `"Environment.GetEnvironmentVariable("SQLSTR")"` will return `SQLSTR's` secret value

BeforeFeature (Invoking webdriver)

- Required: Chrome Driver
The solution will attempt to open a Webdriver.
- (Optional) Tagging the Feature as “@NoBrowser” will prevent the solution from opening a Webdriver.

```
@NoBrowser  
Feature: Algorithms
```

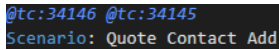
Teardown (AfterTest, AfterFeature, AfterScenario)

A set of static functions that will run as part of “teardown”.

Order of execution is AfterScenario, then AfterFeature, then AfterTest:

- AfterScenario: Runs after every scenario executed.
- AfterFeature: Runs once per Feature (file) post execution of AfterScenario.
- AfterTest: Runs once per execution post execution of AfterScenario.

AfterScenario (Gathering test results)

- The solution will look for the tags starting with “@tc:”

- The solution will save what the result of the scenario was (pass/fail)
- See AfterFeature (Devops result submission) for next steps.

AfterFeature (Devops authentication)

Locally:

- The solution will look for “./SessionToken.temp” (file ignored in .gitignored)
- “SessionToken.temp” should be a json file containing “AZUREDEVOPS_PAT” variable, it’s value should be the PAT used to connect to Azure Devops.



- The PAT will be encrypted and sent as authentication to Devops API.



- Azure Pipeline:
 - Load AZUREDEVOPS_PAT using the AccessToken from the pipeline service provider



AfterFeature (Devops result submission)

- The solution will submit the gathered results from AfterScenario (Gathering test results).
- Using the following Environment variables as parameter

```
"AZUREDEVOPS_ORGANIZATION": "biberk",  
"AZUREDEVOPS_PROJECT": "Apollo",  
"AZUREDEVOPS_TESTPLAN_ID": "10364",  
"AZUREDEVOPS_TESTSUITE_ID": "15133",
```

- The solution gathers “Test Points” and mark those with the outcome gathered in AfterScenario (Gathering test results) (pass/fail).

AfterTest (disposal)

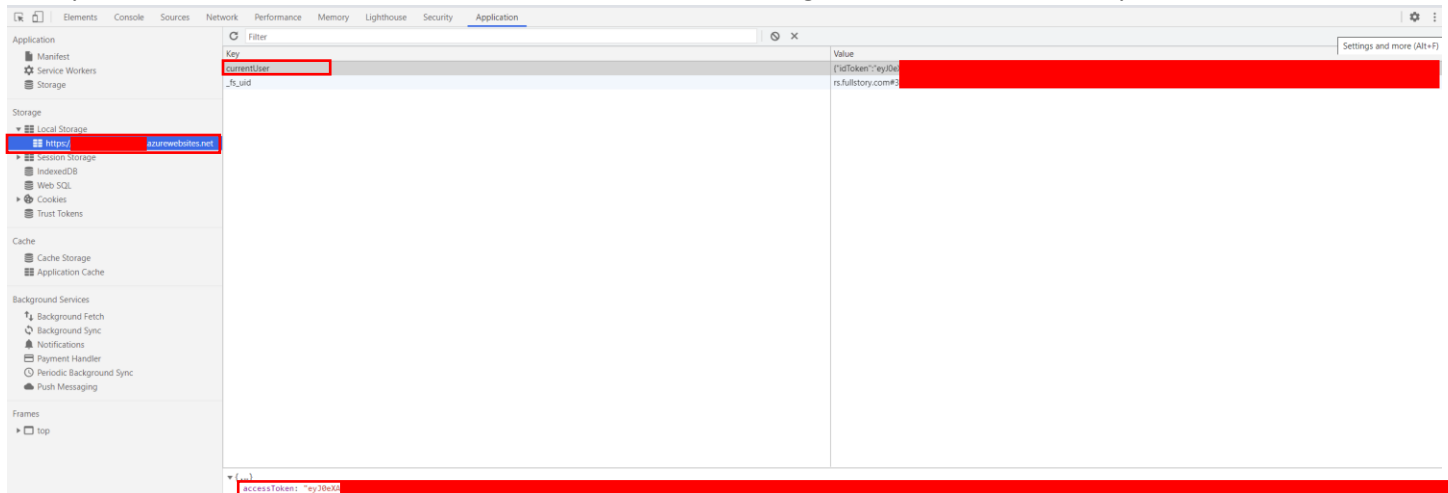
The system to dispose all connections/Webdrivers.

RestAPI

Authentication (from Webdriver)

Condition: if the feature has a Webdriver initiated

The system will look for accessToken in the Webdriver's local storage under "currentUser" key



Authentication (Bearer generation)

Required: provide the following variables in the environment file:

API_TENANT_ID_SECRETNAME
API_CLIENT_ID_SECRETNAME
API_CLIENT_SECRET_SECRETNAME
API_USERNAME_SECRETNAME
API_PASSWORD_SECRETNAME

The solution will attempt to retrieve the bearer token through Microsoft authentication using the above variables.

```
private static String _APIToken = null;

private static String getAuthTokenAPI()
{
    if (_APIToken == null)
    {
        string TenantId = Environment.GetEnvironmentVariable(Environment.GetEnvironmentVariable("API_TENANT_ID_SECRETNAME"));
        string ClientId = Environment.GetEnvironmentVariable(Environment.GetEnvironmentVariable("API_CLIENT_ID_SECRETNAME"));
        string ClientSecret = Environment.GetEnvironmentVariable(Environment.GetEnvironmentVariable("API_CLIENT_SECRET_SECRETNAME"));
        string Username = Environment.GetEnvironmentVariable(Environment.GetEnvironmentVariable("API_USERNAME_SECRETNAME"));
        string Password = Environment.GetEnvironmentVariable(Environment.GetEnvironmentVariable("API_PASSWORD_SECRETNAME"));

        HttpContent content = new FormUrlEncodedContent(new[]
        {
            new KeyValuePair<string, string>("grant_type", "PASSWORD"),
            new KeyValuePair<string, string>("client_id", ClientId),
            new KeyValuePair<string, string>("client_secret", ClientSecret),
            new KeyValuePair<string, string>("scope", $"openid {ClientId}/.default"),
            new KeyValuePair<string, string>("username", Username),
            new KeyValuePair<string, string>("password", Password)
        });

        var response = POST($"https://login.microsoftonline.com/{TenantId}/oauth2/v2.0/token", null, content);

        _APIToken = response["access_token"];
    }

    return _APIToken;
}
```

SQL

Authentication

Required: must provide "SQL_CONNECTIONSTRING_SECRETNAME" containing the secret name of the secret containing the SQL connection string

The system will attempt to connect using the above connection string

```
using (SqlConnection connection = new SqlConnection(Environment.GetEnvironmentVariable(Environment.GetEnvironmentVariable("SQL_CONNECTIONSTRING_SECRETNAME"))))
```

Cosmos

Authentication

Required: must provide “COSMOS_TARGETURL_SECRETNAME” & “COSMOS_APIKEY_SECRETNAME” containing the secret name of the secret containing the Cosmos Endpoint & API Key.

The system will attempt to connect using the above provided values:

```
public static CosmosClient client = new CosmosClient(Environment.GetEnvironmentVariable(Environment.GetEnvironmentVariable("COSMOS_TARGETURL_SECRETNAME")),  
                                                    Environment.GetEnvironmentVariable(Environment.GetEnvironmentVariable("COSMOS_APIKEY_SECRETNAME")));
```

Note: the system will dispose the above client in AfterTest function.